OOABAP

Limitations in Procedural approach

1. Editing report output and save in database tables is not possible
2. Limited events like at selection-screen, start-of-selection
3. Performance is bit week compare to OOPS
4. FM names wrong, not show any error at compile time. At run time it will go to dump.
5. Security (Visibility sections)

Features:

- Encapsulation → Binding entire details into single unit
- Data Abstraction → By using visibility section we can hide the data.
- Inheritance → Reusability in sub classes
- Polymorphism →

In OOABAP, everything we can write in form of Class and Methods and call through object.
Class is the blueprint or template of an object. Object is the real one.

EX: - If you want to build a form house then we take a plan from engineer to build the form house. It's nothing but class. Based on the plan constructed house is an object.

Note: - Based on one class we can create any number of objects.

There are two types of classes.
1. Local Class
2. Global class

Differences between Local & Global classes

Local	Global
1. Local class name starts with any letter	1) Global class name must start with 'Y' or 'Z'.
2. It's created through SE38 transaction.	2) It's created through SE24 transaction.
3. We can access the local class with in the program only.	3) We can access the global class from any where in the SAP.
4. Local class is stored in the memory of ABAP program.	4) Global class is stored in the class repository.

A class contains two sections.
1. Class Definition
2. Class Implementation

A class definition contains components. Components means Attributes, Constants, Methods, Events, Interfaces.

Attributes: - Attributes are used to declare the variables, work areas, internal tables which are needed to implement the logics.
Constants: - Constants will contain fixed value which will not change entire class.
Methods: - Method is the collection of statements which perform the particular activity. Methods are coding blocks of a class, which can provide some business functionality
Events: - Event is an action which is performed at run time. Events are used to provide the dynamic features at run time. Events are used to handle the methods of some other class.
Interface: - Interface is the collection of methods which are defined & not implemented.

Attributes, Methods, Events will contain in two forms.

1. Instance
2. Static

There are three types of visibility sections.
1. Public Section
2. Protected Section
3. Private Section

Public Section: - We can access the public components within the class as well as outside the class.
Protected section: - We can access the protected components within the class as well as derived or child class.
Private section: - We can access the private components within the class only.

Note: - In ABAP we haven't default visibility section.

```
REPORT ZLOCAL_CLASS1.
class lcl_test1 definition.
  PUBLIC SECTION.
  data a type i.
  class-data b type i.
  CONSTANTS c type i value 30.
  ENDCLASS.

*  Write:/ a.   "Syntax error
*  Write:/ b.   "Syntax error
*  Write:/ c.   "Syntax error
*   write:/ lcl_test1->a.   "Syntax error
*   write:/ lcl_test1=>a.   "Syntax error
*   write:/ lcl_test1->b.   "Syntax error
    write:/ lcl_test1=>b.
*    write:/ lcl_test1->c. "Syntax error
    write:/ lcl_test1=>c.
```

Component which is starting with 'Data' is static (attribute or method or event) and which is starting with 'Class-data' is Static.

Instance or Static components we can't use directly outside of the class. By using class name, we can use Static components outside of the class and by using object we can use instance or static components outside of the class.

Instance variable and Static variable how work in background.

```
CLASS lcl_test1 DEFINITION.
  PUBLIC SECTION.
    DATA a TYPE i.
    CLASS-DATA b TYPE i.
ENDCLASS.

DATA obj1 TYPE REF TO lcl_test1.
DATA obj2 TYPE REF TO lcl_test1.
CREATE OBJECT obj1.
CREATE OBJECT obj2.
WRITE:/ obj1->a, obj1->b.

ULINE.
obj1->a = 10.
obj1->b = 20.
WRITE:/ ' Assign values to a, b from OBJ1'.
```

```
WRITE:/ obj1->a, obj1->b.
ULINE.
WRITE:/ 'Just calling a, b from OBJ2'.
WRITE:/ obj2->a, obj2->b.
ULINE.
WRITE:/ 'Assign values to a, b from OBJ2'.
obj2->a = 89.
obj2->b = 90.
WRITE:/ obj2->a, obj2->b.
ULINE.
WRITE:/ 'Display a, b from OBJ1'.
WRITE:/ obj1->a, obj1->b.
ULINE.
```

Output:

0 0	**1**
Assign values to a, b from OBJ1 10 20	**2**
Just calling a, b from OBJ2 0 20	**3**
Assign values to a, b from OBJ2 89 90	**4**
Display a, b from OBJ1 10 90	**5**

Initially the value for a, b will be 0 (default value for integer is 0). So in the 1st case 0, 0 is coming in the output.
Then I'm initializing the values to a = 10 and b = 20. Same values are coming in the 2nd case.
First two cases I'm calling from OBJ1.
I've created one more Object named as OBJ2. From OBJ2 calling a, b attributes.
The values will be 0, 20 is coming.

Instance variable will create separate memory for new objects. For example, I've created 10 objects for a class. At that time, 10 memory locations will create for Instance variable.
Static variable will create single memory in life time of class. For example, I've created 10 objects for a class. Only one memory location will create for Static variable.

From OBJ1 I'm assigning something value to Static variable. The same value should pick by all other objects. Instance variable value will be specific for object.

In 4th case, I'm assigning values a = 89, b = 90 from OBJ2 and calling those variables. Values are displaying as expected.

In 5th case, I'm calling a, b from OBJ1. From OB1, I've updated instance variable updated as 10 in 2nd case. System will pick that value. For static variable 'b', last time updated in case 4. System will pick that value.

Methods calling which are from Private or Protected Sections

```
CLASS lcl_test1 DEFINITION.
  PUBLIC SECTION.
    DATA a TYPE i.
    DATA b TYPE i.
```

```
    METHODS get_data IMPORTING i_a TYPE i OPTIONAL
                               i_b TYPE i.
  PROTECTED SECTION.
    METHODS display_data.
ENDCLASS.

CLASS lcl_test1 IMPLEMENTATION.
  METHOD get_data.
    a = i_a.
    b = i_b.
  ENDMETHOD.

  METHOD display_data.
    WRITE:/ a, b.
  ENDMETHOD.
ENDCLASS.
DATA obj1 TYPE REF TO lcl_test1.
START-OF-SELECTION.
  CREATE OBJECT obj1.
  PARAMETERS: P_a TYPE i, p_b TYPE i.
  CALL METHOD obj1->get_data
    EXPORTING
      i_a = p_a
      i_b = p_b.
  CALL METHOD obj1->display_data.
```

Program is not activating. Because we can't call the Private or Protected methods outside of the program. Don't call Private or Protected methods externally. We can call these methods internally (Method calling with in another method).

```
  METHOD get_data.
    a = i_a.
    b = i_b.
    call method me->display_data.
  ENDMETHOD.
```

In method GET_DATA implementation I'm calling one more method (Private or Protected method). System will not show any error now.
Here I've used 'ME->DISPLAY_DATA'. 'ME' means current object. If we are calling this method from OBJ1 then in place of ME, OBJ1 will come internally.

I've mentioned in method declaration as below.

```
METHODS get_data IMPORTING i_a TYPE i OPTIONAL
                           i_b TYPE i.
```

Here Importing I'm mentioning. If we want to import something value from outside then we need to go for Importing parameters. `i_a TYPE i OPTIONAL` means, i_a is optional value. While calling this method from outside program, if we pass something value to i_a, it will pick that value otherwise it will pick the default value as 0.
For i_b, I'm not maintaining the key work as 'OPTIONAL' means, we need to pass something value to this variable while calling this method from outside program. Otherwise system will through error message.

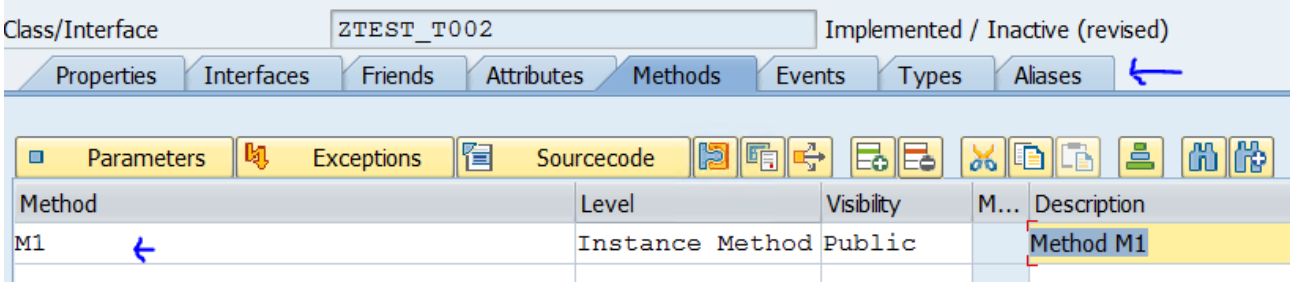
```
PROTECTED SECTION.
    METHODS display_data.
```

Protected Section or Private Section methods we can't use outside of the class. If we keep the method in Public Section, then only we can call the method outside of the class.

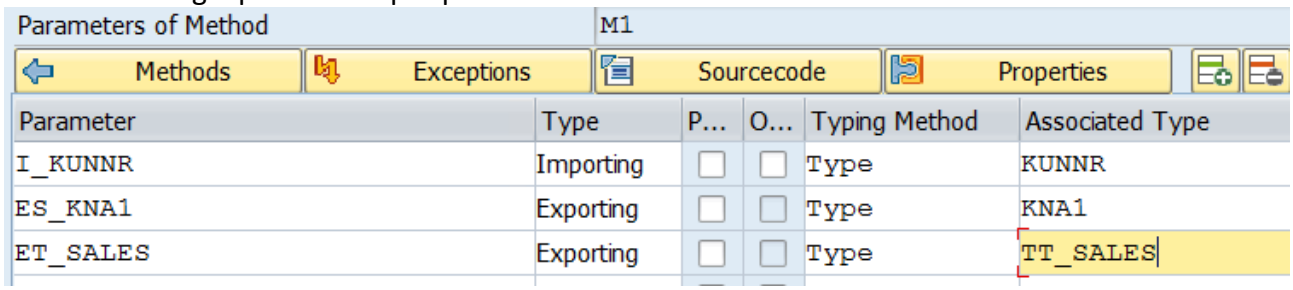
Creating Global class

Execute **SE24** transaction. Provide the class name, click on create button. Two radio buttons will come as class and interface. Select 'class' radio button click on enter button. Provide the

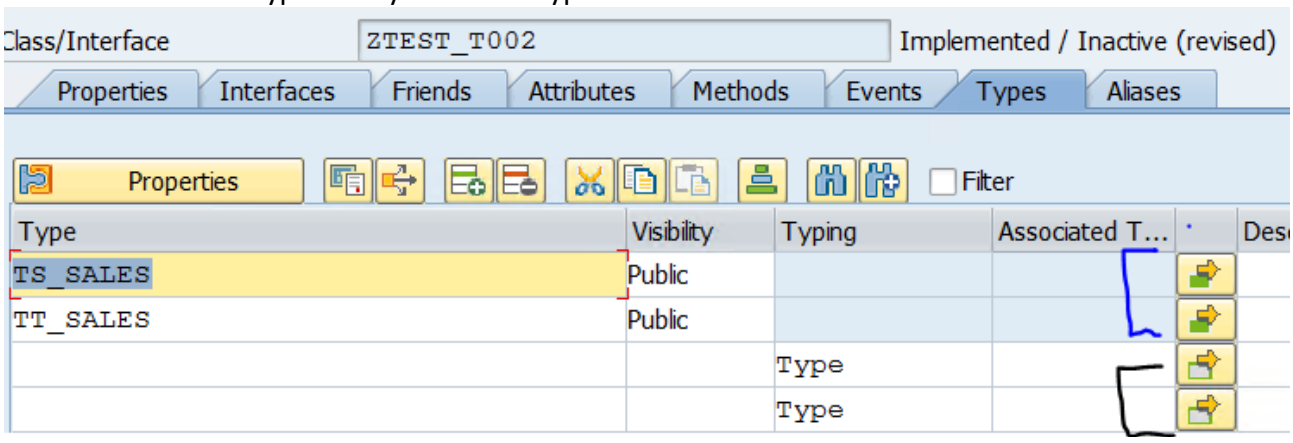
description of the class. We can see Instant Generation as 'PUBLIC'. Click on Enter button. Provide package, click on save button.
Now we can see the screen with a few tabs like Properties, Interfaces, Friends, Attributes, Methods, Events, Types, Aliases. By default Methods tab will open. We can declare Methods here.



We can see Method, Level, Visibility, Description here. Method means, method name we need to mention, Level means what type of method it is like Instance or Static method. Visibility we can select as Public, Protect, Private and we can maintain Description of method under Description. We can see Parameters, exceptions, source code buttons also here. If we want provide any input parameters or output parameters or changing parameters or returning parameters then we can provide by clicking on Parameters button. By using 'Exceptions' button, we can add exception to the method. By using 'Sourcecode' button, we can implement the method.
Give something input and output parameters as below. Click on 'Parameters' button.



The naming convention I'm giving like I_KUNNR means Importing Customer number and ES_KNA1 means, Exporting structure (work area) from KNA1 table, ET_SALES means Exporting Table (Internal table). For I_KUNNR I've taken Associated type as 'KUNNR' and for ES_KNA1 I've taken associated type as 'KNA1' total table structure will come as reference. If we want to declare internal table, then we need to declare table type in SE11 transaction (Global table type) or Locally (in side the Class-Types tab). Here I've mentioned Associated type as 'TT_SALES' for ET_SALES. I've declared one table type locally. Click on 'Types' tab.

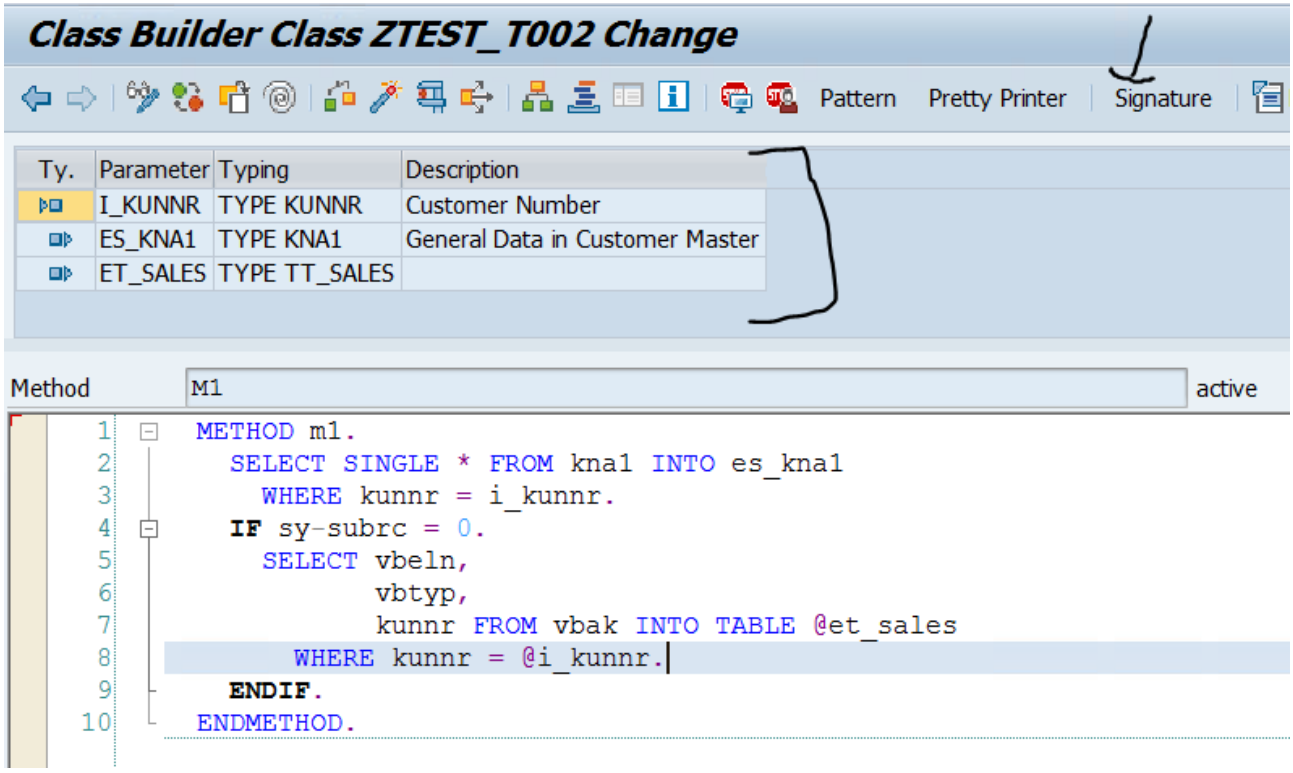


I've declared Type structure first based on that one, I've declared Table Type. Provide the Type Structure name, visibility as 'Public', click on ' ➔ ' button as shown in above image. If we have any global structure we can that name otherwise click on arrow button.

```
public section.

  types:
```

```
    BEGIN OF ts_sales,
       vbeln type vbak-vbeln,
       vbtyp type vbak-vbtyp,
       kunnr type vbak-kunnr,
  end of ts_sales .
  types:
    TT_SALES type table of ts_sales .
```

Write the code as above. Click on Save button, activate, click on back button.
Now we can see TS_SALES and TT_SALES. By using this TT_SALES table type, I've declare the exporting parameters in Method 'M1'.
Click on 'Methods' tab, select the method, click on 'Source code' button.



We can see Importing, exporting parameters here. If we are not able see these parameters, click on 'Signature' button. Then we can see these parameters.
Implement the logic as below.

Now if we want to call this method, then need create one local program there we can call the method.

```
PARAMETERS p_kunnr TYPE kunnr.
DATA obj TYPE REF TO ztest_t002.

START-OF-SELECTION.
  CREATE OBJECT obj.

  CALL METHOD obj->m1
    EXPORTING
      i_kunnr   = p_kunnr                    " Customer Number
    IMPORTING
      es_kna1   = DATA(ls_kna1)                    " General Data in Customer Master
      et_sales  = DATA(lt_sales).
  LOOP AT lt_sales INTO DATA(ls_sales).
    WRITE:/ ls_sales-vbeln, ls_sales-vbtyp, ls_sales-kunnr.
  ENDLOOP.
```

Parameters I am declaring, Object declaring, creating the object.
Now calling the method. Provide like 'CALL METHOD OBJ->' and click on CTRL + Space button. It will show list of the methods available under that class. Select the method. Click on CTRL + Space

button. Click on Shift + Enter button. It will display the Input and Output parameters if the method contains.
Provide Importing, Exporting parameters. Save, check, activate the program, click on execute button.
Note:- Importing parameters value we can not change in the method or in debugging.
If we want to change the importing parameter in method implementation, instead of taking 'Importing', need to take 'Changing' option. Changing will work like Import as well as Export also (both functionalities).

Ty.	Parameter	Typing	Description
	I_COUNTRY	TYPE LAND1	Country Key
	ET_T001	TYPE TT_T001	Company Codes
	C_COUNTRY	TYPE LAND1	Country Key

Method GET_COMP active

```
method GET_COMP.
  data lv_bukrs type bukrs.
  data lv_country type land1.

  select single bukrs from t001 into lv_bukrs where land1 = i_country.
    if lv_bukrs is INITIAL.
      c_country = 'DE'.
      endif.
  select bukrs butxt land1 from t001 into table et_t001 where land1 = c_country.

    CALL METHOD ME->GET_VENDOR.
endmethod.
```

In the above image, we can see the example of Changing parameter.

Inheritance:

Inheritance is used to give parent and child relationship in between of classes.
Create a Class with 3 methods (Public, Protect, Private) as below. While creating the method, do not select the 'Final' checkbox.

Create Class ZCL_INHRTC1

Class	ZCL_INHRTC1
Description	testing
Inst.Generation	Public

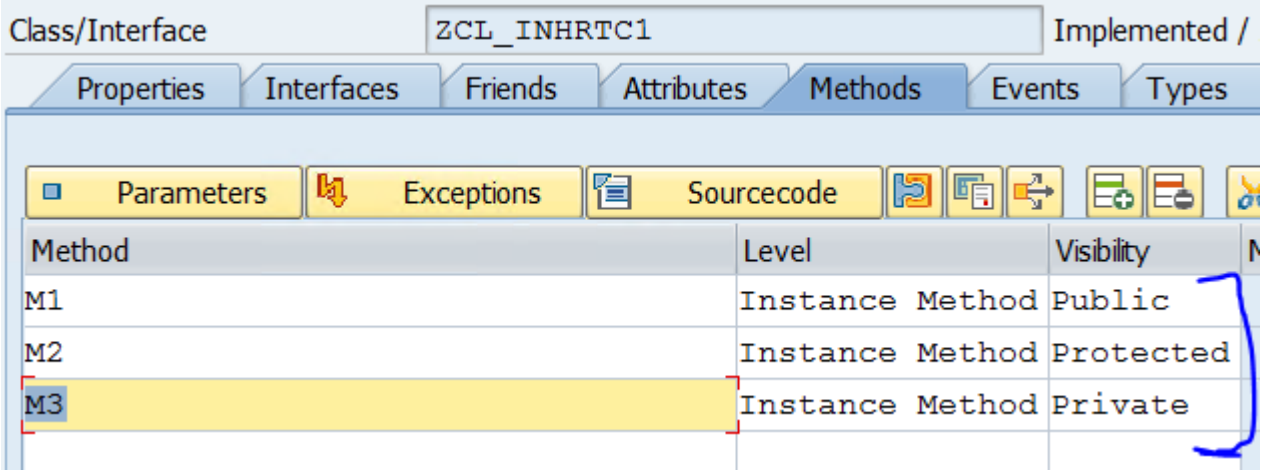
Class Type

- (•) Usual ABAP Class
- ( ) Exception Class
  - [x] With messages of message classes as exceptio
- ( ) Persistent class
- ( ) Test Class (ABAP Unit)

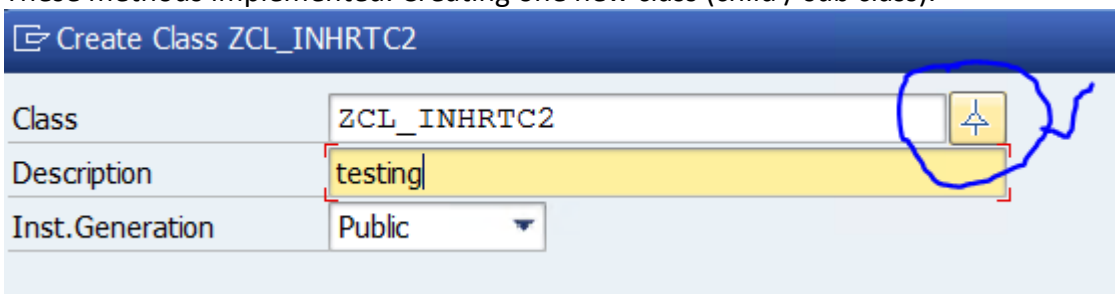
[ ] Final

Methods I can create like this.

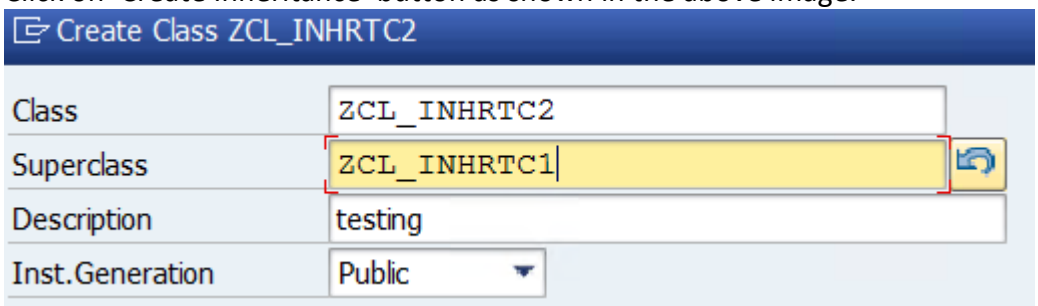
Note: Final Class will not support Inheritance concept.



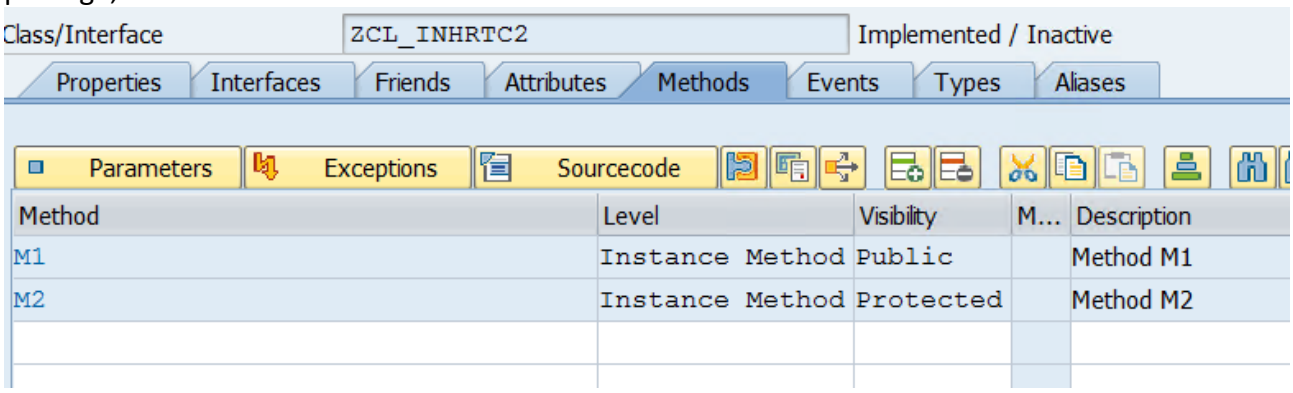
These methods implemented. Creating one new class (child / sub class).



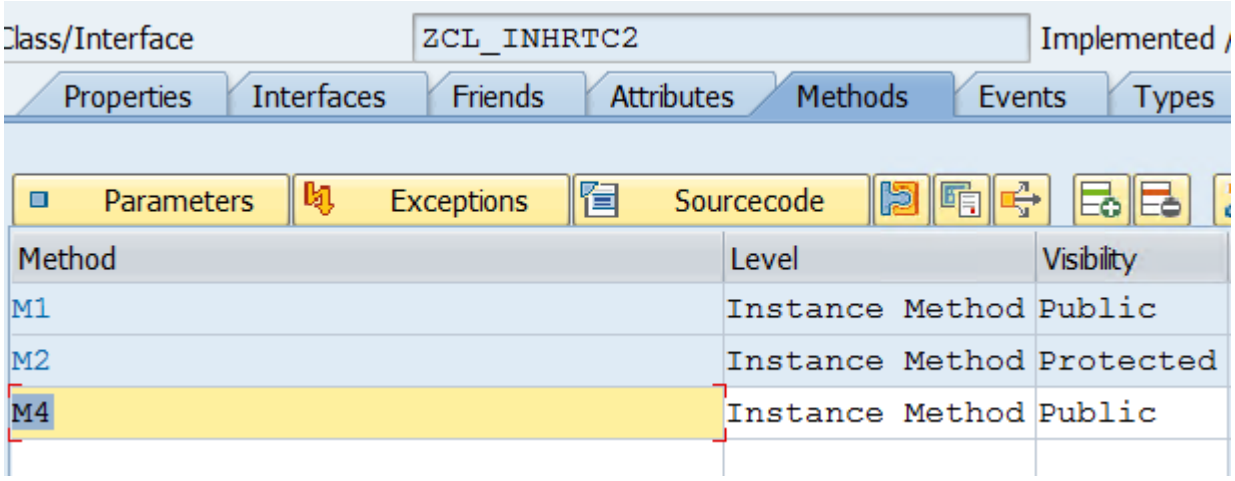
Click on 'Create Inheritance' button as shown in the above image.



Now system will ask the Super class name. Provide super class name, click on 'OK' button, provide package, save the class.

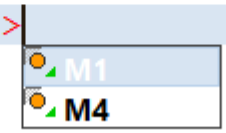


Automatically Methods will come from Super class. Here we can see M1 and M2 methods only. M3 is not coming here because M3 method is available in Private Section. Private Section Methods can't use outside of the class (even though in child class).
I want to add one new method in child class.



Now I'll create object / Instance for child class and call the methods.

```
REPORT ZTEST_2899.

data obj type ref to ZCL_INHRTC2.
start-OF-SELECTION.
create object obj.

call method obj->
```

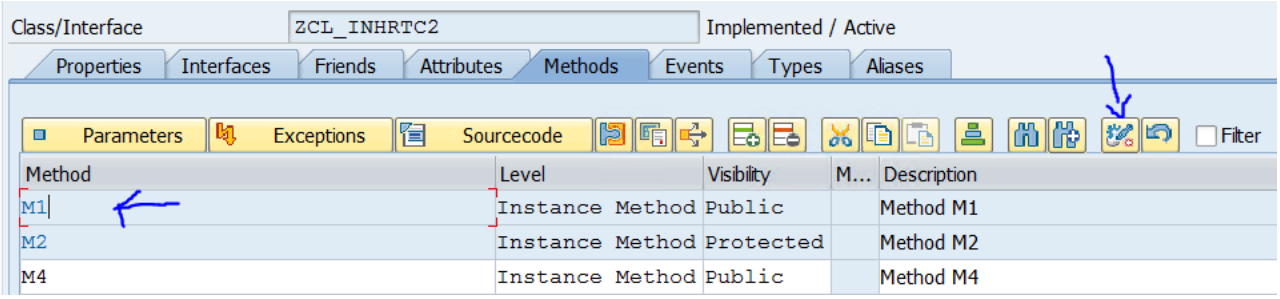


I'm calling methods. So written the syntax like 'CALL METHOD OBJ->' and I have given CTRL + Space button. It is showing list of methods under super and sub classes. Method 'M2' is not showing here. Because method 'M2' is in Protected Section. Protected Section Methods can be used with in the class or subclass only. Outside of class or outside of subclass we can't use it. That is why it is not showing the 'M2' method.

```
data obj type ref to ZCL_INHRTC2.
start-OF-SELECTION.
create object obj.

call method obj->m1.
call method obj->m4.
```

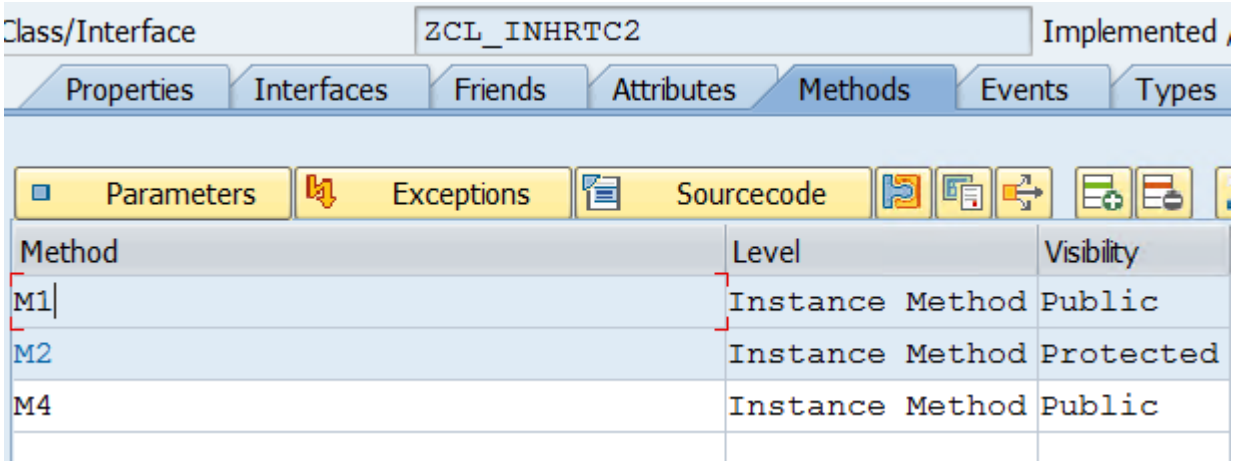
Here system will not throw any error. Because Child class object / instance can access all Super class public methods.

Method 'M1' is created in Super class. Now I want to define same method in Child class also.



Method 'M1' and 'M2' background colours are in different colour and text colour also different compare with 'M4' method. It means 'M1' and 'M2' are not the current class methods. Only 'M4'

method is in editable mode and there is not background colour. 'M4' method is available in the current class.
I'm defining 'M1' method in sub class also.
Select the method, click on 'Redefine Method' button as shown in the above image.



Now method 'M1' text colour changed to black colour but background colour didn't change. Means, Method 'M1' is redefined in this sub class.

If we call method M1 from program then it will call from sub class only. If we want to call Super class 'M1' method from sub class object, need to write the logic like this in child class method implementation.

```
METHOD m1.
  CALL METHOD super->m1.
  WRITE:/ 'Method M1 from ZCL_INHRTC2' .
ENDMETHOD.
```

By using 'CALL METHOD SUPER->M1', System will call the super class method.

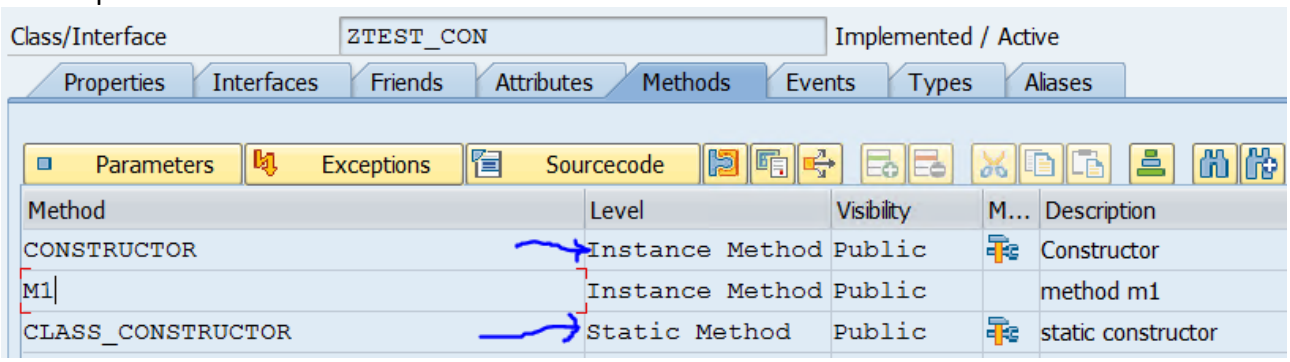
Convert local class to Global class

Execute SE24 transaction. Click on 'Object Type' menu button → click on 'Import' → click on 'Local classes in Program' → Provide the program name (Local contain program name). Click on enter button. Provide Global class name, select that row, click on Import button (F5). Provide the package. We can see one success message. Open the Global class activate it.

Constructor:

Constructor is also one method only. But it is special method. The speciality is this Constructor method will call automatically after creating object, not required to call it externally.

Method name we need to take as 'CONSTRUCTOR' only. System can automatically understand like this is special method.



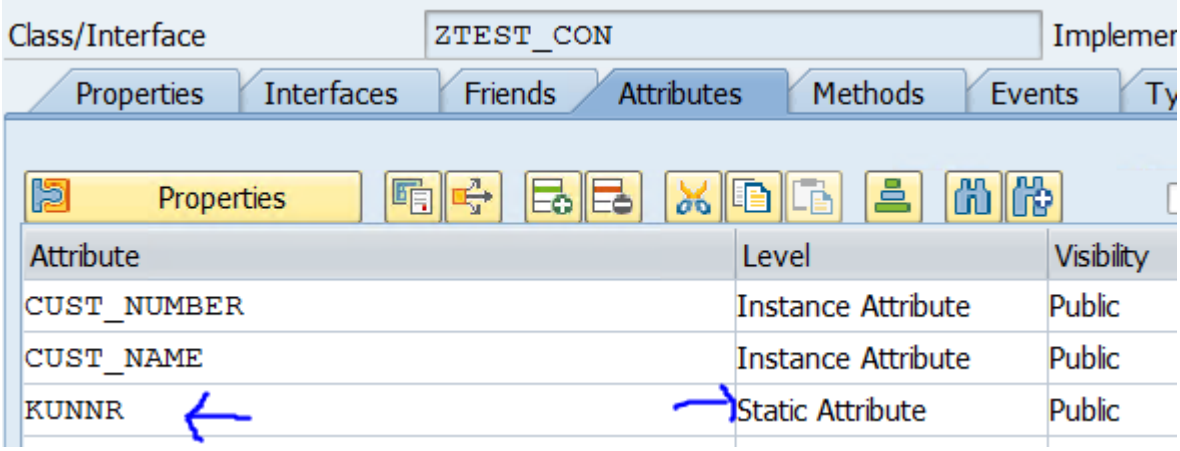
I've taken 3 methods here.
Constructor is a special method and it is Instance Method. CLASS_CONSTRUCTOR is also a special method and it is Static Method. 'M1' is normal method.
Let us create one object for this class in a program.

```
data obj1 type ref to ZTEST_CON.
start-OF-SELECTION.
create object obj1.
```

I'm just creating the object for class and execute the program.

```
Static Constructor from ZTEST_CON
Instase Constructor from ZTEST_CON
```

Here Static Constructor is calling first and Instance constructor is calling next.
I'm not calling any method, just created object. At that time only these two special methods are calling.
I'll do one thing here.
I don't create any object & I'll take one static attribute in class and assign something value from program.



I've mentioned 'KUNNR' as static attribute. Now I'll assign something value to this attribute from program like this.

```
REPORT ZTEST_CON_123.
*data obj1 type ref to ZTEST_CON.
*start-OF-SELECTION.
*create object obj1.

ztest_con=>kunnr = '123'.
```

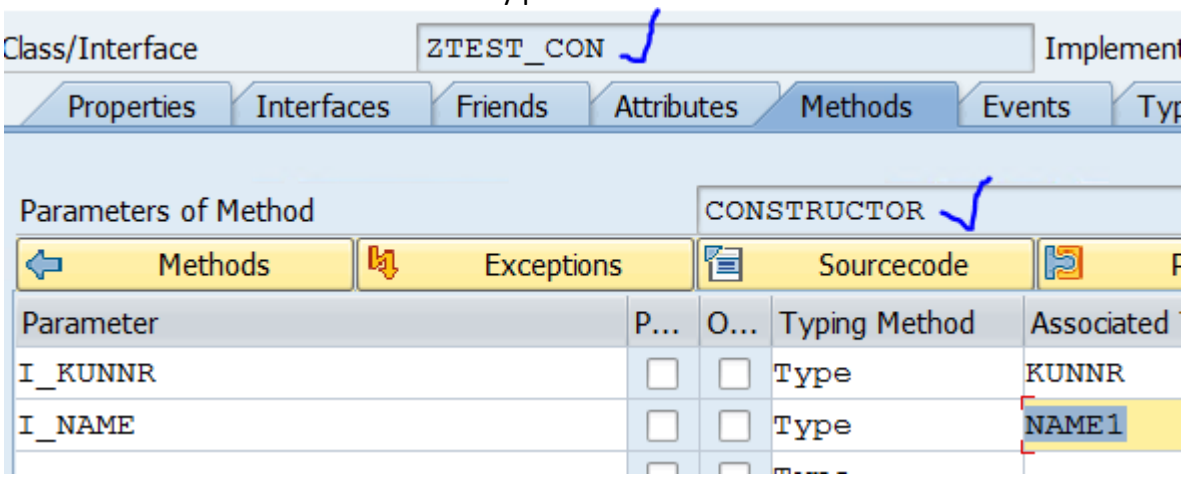
I'm just assigning static attribute value and click on execute button.

```
testing
---------------------------------------
Static Constructor from ZTEST_CON
```

Automatically Static Constructor is calling internally. So Static constructor can trigger in two ways.
If we are assigning something value to static attribute it will trigger and while creating the object static constructor will trigger. Only one time it will trigger.
But coming Instance constructor, It will trigger at the time of creating object only.

Constructor is triggering automatically then how can we pass input and output parameters?

Instance Constructor can accept Input parameters only. It will not accept any other parameters. Static Constructor should not allow any parameters.



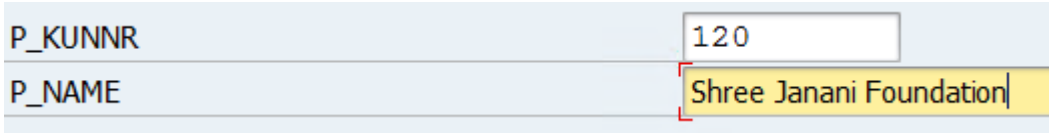
I've given the input parameters to Instance Constructor.

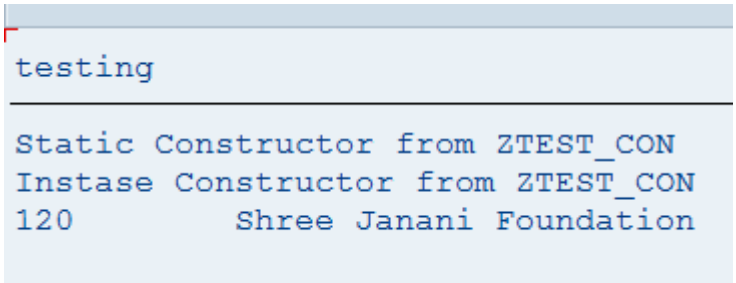
```
  method CONSTRUCTOR.
    cust_number = i_kunnr.
    cust_name = i_name.
    write:/ 'Instase Constructor from ZTEST_CON'.
    write:/ cust_number, cust_name.
  endmethod.
```

Logic I've written inside the Instance Constructor. Now I'll pass the values to constructor from program.

```
data obj1 type ref to ZTEST_CON.
PARAMETERS: p_kunnr type kunnr,
            p_name type name1.
start-OF-SELECTION.
create object obj1 EXPORTING i_kunnr = p_kunnr
                             i_name = p_name.
```

If I execute the program, both special methods will trigger. Output will come like this.





First it is triggering Static Constructor and next Instance constructor is triggering.

As of now I'm removing the input parameters to the Instance constructor and I'm creating more than one object for the class.

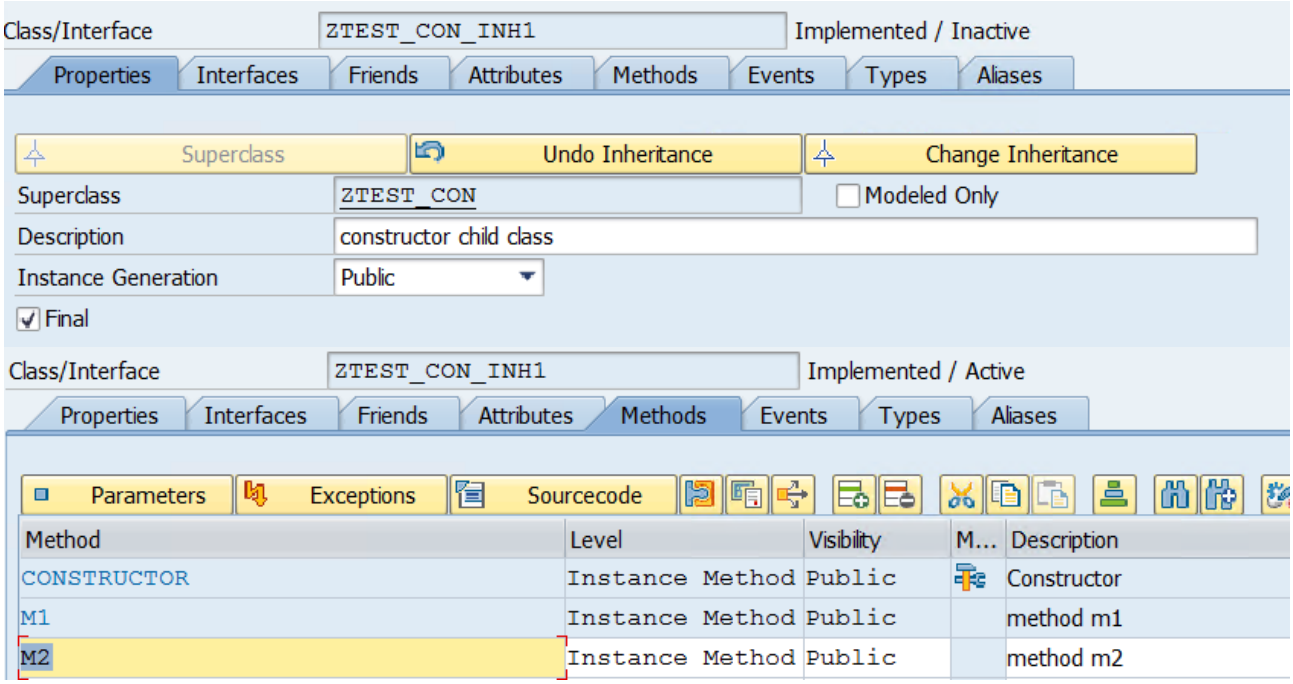
```
REPORT ZTEST_CON_123.
data obj1 type ref to ZTEST_CON.
data obj2 type ref to ZTEST_CON.
data obj3 type ref to ZTEST_CON.
start-OF-SELECTION.
create object obj1.
create object obj2.
create object obj3.
```

Output will come like this.

```
testing

Static Constructor from ZTEST_CON
Instase Constructor from ZTEST_CON
Instase Constructor from ZTEST_CON
Instase Constructor from ZTEST_CON
```

Only one-time Static constructor is triggering and three times Instance constructor is triggering. Instance Constructor is specific for Object. How many number objects created, that many of times instance constructor will trigger. But Static constructor will trigger only once in life time of the class.

I'll take one child class for this class and create object for child class.



Automatically constructor, Method M1 is coming from super class.
Method M2 I've taken in the child class & implemented as below.

```
method M2.
  write:/ 'Method M2 from ZTEST_CON_INH1 Class'.
endmethod.
```

Now I'll create a program.

```
REPORT ZTEST_28731.
data obj1 type ref to ZTEST_CON_INH1.
start-OF-SELECTION.
create object obj1.
```

Just Object is created for child class. Execute the program.

```
testing

Static Constructor from ZTEST_CON
Instase Constructor from ZTEST_CON
```

Automatically super class Static and Instance constructors are calling because child class object can access the super class methods.

Let me add one Instance Constructor in the child class.

Class Builder: Change Class ZTEST_CON_INH1

Local Definitions/Implementations | Local Test Classes

Class/Interface: ZTEST_CON_INH1 — Implemented / Inactive

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

Parameters | Exceptions | Sourcecode

Method	Level	Visibility	M...	Description
M1	Instance Method	Public		method m1
M2	Instance Method	Public		method m2
CONSTRUCTOR	Instance Method	Public		Constructor

1 Syntax Error for Class ZTEST_CON_INH1, Method CONSTRUCTOR

Type	Line	Description
	2	Method CONSTRUCTOR
		The constructor method must call the constructor of the superclass (SUPER->CONSTRUCTOR).

It is showing something error. If we are maintaining the Instance Constructor in Super class as well as child class, then we need to call super class constructor externally in child class Instance constructor implementation.

Class Builder Class ZTEST_CON_INH1 Change

Pattern | Pretty Printer | Signature

Ty. | Parameter | Typing | Description

Method: CONSTRUCTOR — active

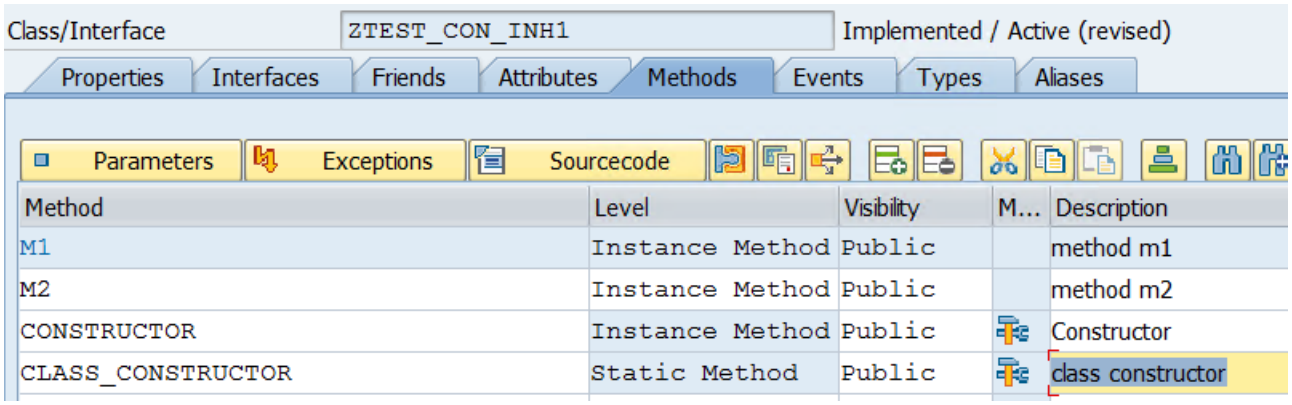
```
1  method CONSTRUCTOR.
2    write:/ 'Method CONSTRUCTOR from class - ZTEST_CON_INH1'.
3    CALL METHOD SUPER->CONSTRUCTOR.
4  endmethod.
```

Now it is not showing any error.
Execute the program once again.

```
testing
---------------------------------------------------
Static Constructor from ZTEST_CON
Method CONSTRUCTOR from class - ZTEST_CON_INH1
Instase Constructor from ZTEST_CON
```

First Super class Static Constructor is triggering, then Sub class Instance Constructor is triggering, then super class Instance constructor is triggering.

I'll add Static Constructor in child class.



Implemented static constructor as below.

```
method CLASS_CONSTRUCTOR.
  write:/ 'Static Constructor from class ZTEST_CON_INH1'.
endmethod.
```

Execute the program once again.

```
testing
---------------------------------------------------
Static Constructor from ZTEST_CON
Static Constructor from class ZTEST_CON_INH1
Method CONSTRUCTOR from class - ZTEST_CON_INH1
Instase Constructor from ZTEST_CON
```

Execute sequence:

1. Super class Static Constructor
2. Child class Static Constructor
3. Child class Instance Constructor
4. Super class Instance Constructor

Differences between normal method and special method (constructor).

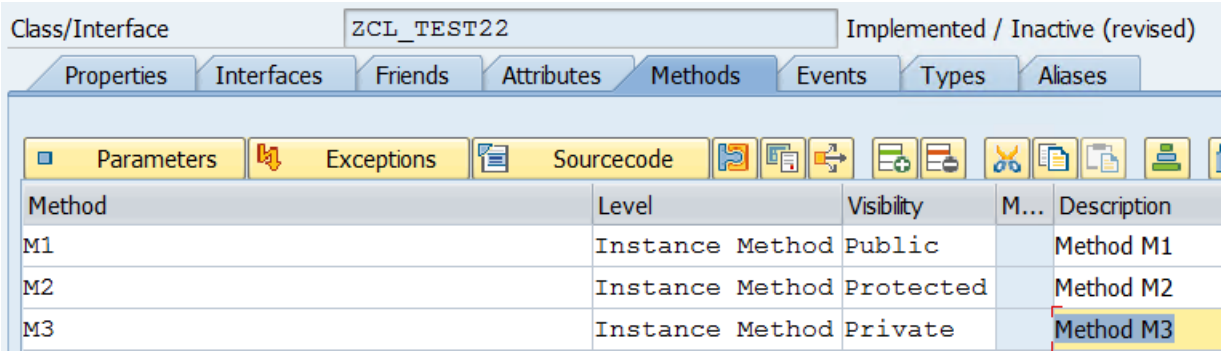
Normal Method	Constructor
It can declare any of section (Public, Protected, Private)	It can declare in only public section
It should call explicitly	Automatically it will call. Explicit call not required
It can any number of times by using the same object	Instance Constructor will call in life time of object. Static Constructor will call in life time of class.
It can contain any type of parameters (Import, export, changing, returning)	Instance Constructor will allow only import parameters, Static Constructors will not allow any parameters
Can return any number of values	Never written any value(exporting values)

Differences between Instance and Static Constructor

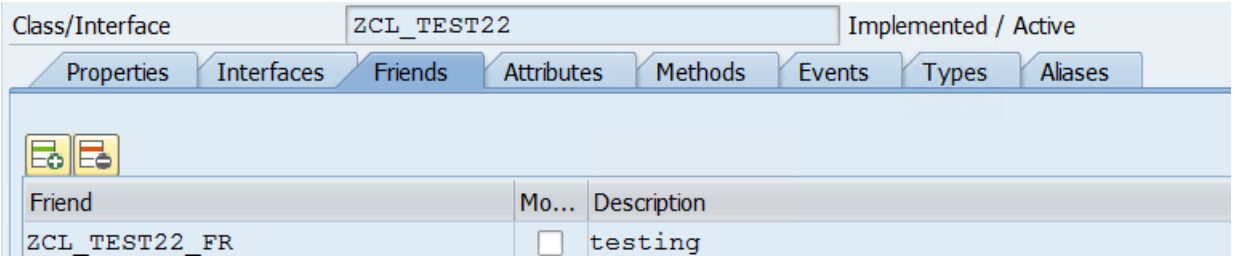
Instance Constructor	Static Constructor
It will allow only input parameters	It will not allow any parameters
Automatically it will execute at the time of object created for the class	Automatically it will execute at the time of object created for the class or assigning something value to the static variable of the class
Specific to object (if we create 10 objects, Instance constructor will call 10 times)	Specific to class (if we create 10 objects, Static constructor will call only 1 time)
Execution Order: 1. Static Constructor 2. Instance Constructor	Execution Order: 1. Static Constructor 2. Instance Constructor

Friend Class

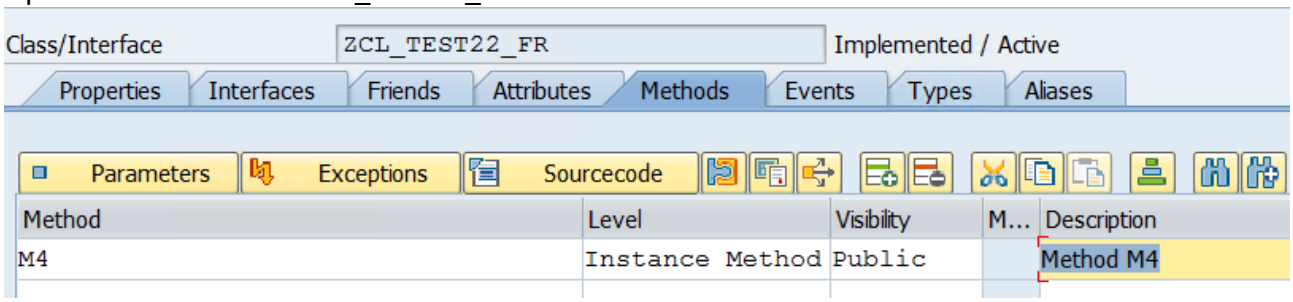
I've created class like this



3 methods I've mentioned with Public, Protected, Private sections. In Friends Tab, I'm giving one class name like this.



Open the friend class – ZCL_TEST22_FR.



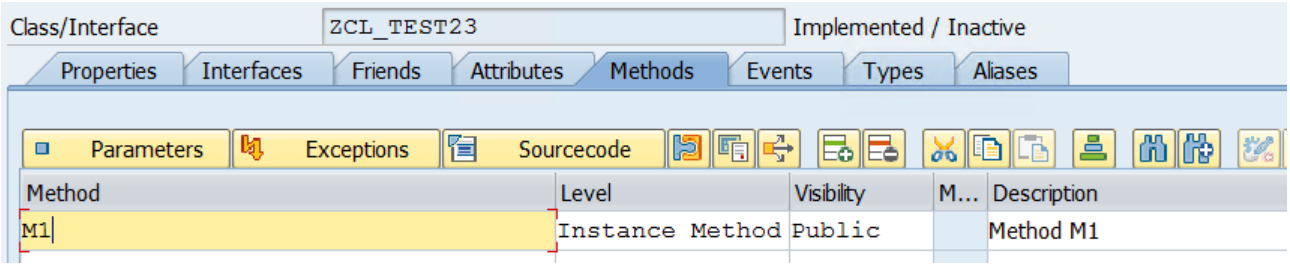
Implemented the method M4 as below.

```
  method M4.
    data obj type ref to zcl_test22.
    create object obj.
    call method obj->m1.
    call method obj->m2.
    call method obj->m3.
  endmethod.
```

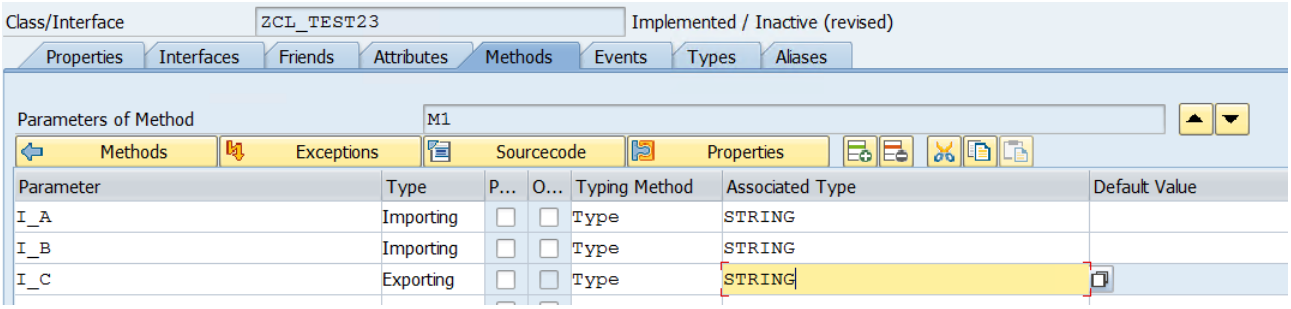
It will not through any error. The class – ZCL_TEST22 is containing Public, Protected and Private methods. Protected and Private methods we can not use outside of the class. But by using friend class, we can use any type of components in friend class.

Exceptions:

Create a class like this



Take importing, exporting parameters for method M1.



Implement the method like this

```
  METHOD m1.
    i_c = i_a / i_b.
  ENDMETHOD.
```

Create a program to pass the values.

```
REPORT ZTEST_28731.
data obj type ref to ZCL_TEST23.
parameters: p_a type string,
            p_b type string.
start-OF-SELECTION.
create object obj.

call method obj->m1
  EXPORTING
    i_a = p_a
    i_b = p_b
  IMPORTING
    i_c = data(lv_c)   .

write:/ lv_c.
```

Execute the program and provide input parameters.

P_A	20
P_B	2

We can the output as '10'.

If we provide input like this

P_A	20
P_B	0

Execute the program.

```
Category                ABAP programming error
Runtime Errors          BCD_ZERODIVIDE
Except.                 CX_SY_ZERODIVIDE
ABAP Program            ZCL_TEST23====================CP
Application Component   Not assigned
Date and Time           19.03.2023 21:25:59 (UTC)

Short Text
    Division by 0 (type P) in program "ZCL_TEST23========
```

System will through dump like this. We can divide any value with '0'. If we divide with '0', it will go to dump. We can check all dumps in ST22 transaction.

This is called Exception. We need to handle the exceptions in method implementation level or method calling time. We can see the exception class in the dump. Copy that exception class name.

Go to method declaration part, click on exception button, provide the exception class name.

Class/Interface ZCL_TEST23 Implemented / Active (revised)

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

Exceptions of Method M1

Methods | Parameters | Sourcecode | Properties

Exception	Resumable	Description
CX_SY_ZERODIVIDE	☐	System Exception Involving Division by Zero

Go to method implementation.

```
METHOD m1.
  DATA obj_ex TYPE REF TO cx_sy_zerodivide.
  TRY.
      i_c = i_a / i_b.
    CATCH cx_sy_zerodivide INTO obj_ex.
      CALL METHOD obj_ex->get_longtext
        RECEIVING
          result = DATA(lv_result).
      WRITE:/ lv_result.
  ENDTRY.
ENDMETHOD.
```

I'm just using try and end try blocks.

If any exception raised, then it will handle by 'Catch' block. In catch block, I'm calling one method from CX_SY_ZERODIVIDE class. It will give the actual message.

Execute program by giving p_b as '0'. Output will come like this

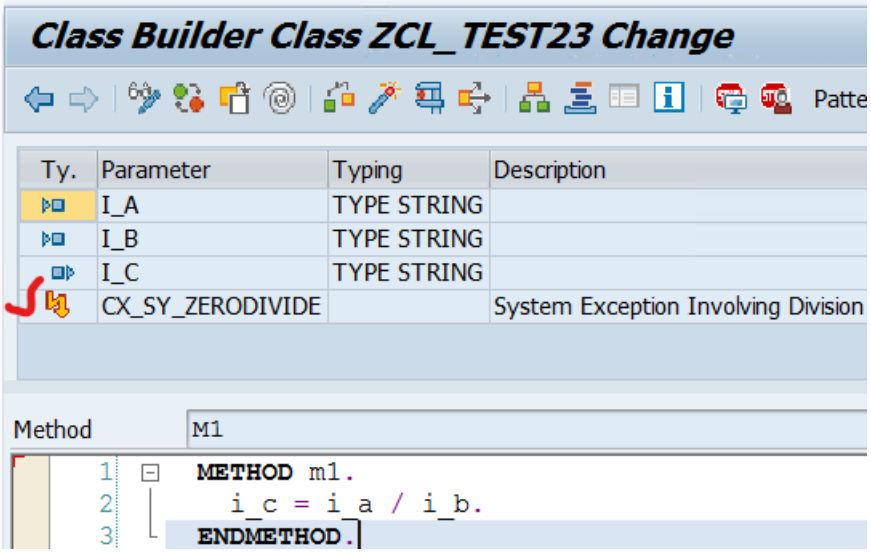
```
testing
---------------------------------------------------------------
You tried to divide by zero during the operation '/'.
```

We have another methods from CX_SY_ZERODIVIDE class.

1. GET_LONGTEXT
2. GET_TEXT
3. GET_SOURCE_POSITION

Each method will give some message .

If we want to handle the exception at time of calling the method, we can do like this.
In method implementation part, remove try and catch blocks as below.



Exception class name will be there in method declaration but Try and Catch blocks removed from method implementation.

Handle the exceptions at the time of method implementation.

```
 REPORT ztest_28731.
DATA obj TYPE REF TO zcl_test23.
DATA obj_ex TYPE REF TO cx_sy_zerodivide.
PARAMETERS: p_a TYPE string,
            p_b TYPE string.

START-OF-SELECTION.
  CREATE OBJECT obj.

  TRY.
      CALL METHOD obj->m1
        EXPORTING
          i_a = p_a
          i_b = p_b
        IMPORTING
          i_c = DATA(lv_c).
    CATCH cx_sy_zerodivide INTO obj_ex.
      CALL METHOD obj_ex->get_text
        RECEIVING
          result = DATA(lv_result).
```

```
      CALL METHOD obj_ex->get_longtext
        RECEIVING
          result = DATA(lv_result1).

      CALL METHOD obj_ex->get_source_position
        IMPORTING
          program_name = DATA(lv_prog)              " ABAP Program: Current
Master Program
          source_line  = DATA(lv_line).

      WRITE:/ 'By using get_text method message -', lv_result.
      WRITE:/ 'By using get_longtext method message -', lv_result1.
      WRITE:/ 'exception is coming in program', lv_prog, 'at line number',
lv_line.
  ENDTRY.
```

Execute the program by giving input p_b = '0'.

```
testing

By using get_text method message - Division by zero
By using get_longtext method message - You tried to divide by zero during the operation '/'.
exception is coming in program ZCL_TEST23====================CP         at line number          2
```

Exception is handling at the method calling level.

Now I'll execute the program by using p_b = 'abc'. The number should not divide by letters. It will go to dump. Execute the program.

```
Category                ABAP programming error
Runtime Errors          CONVT_NO_NUMBER
Except.                 CX_SY_CONVERSION_NO_NUMBER
ABAP Program            ZCL_TEST23====================CP
Application Component   Not assigned
Date and Time           19.03.2023 21:54:39 (UTC)

Short Text
    "ABC" cannot be interpreted as a number
```

We are not handling the exception for dividing with letters. We need to add exception class in method declaration level.

Class/Interface ZCL_TEST23 Implemented / Active

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

Exceptions of Method M1

Methods | Parameters | Sourcecode | Properties

Exception	Resumable	Description
CX_SY_ZERODIVIDE	☐	System Exception Involving Division by Zero
CX_SY_CONVERSION_NO_NUMBER	☐	System exception in transformation to a number

Now handle the exception in method calling.

```
REPORT ztest_28731.
DATA obj TYPE REF TO zcl_test23.
DATA obj_ex TYPE REF TO cx_sy_zerodivide.
```

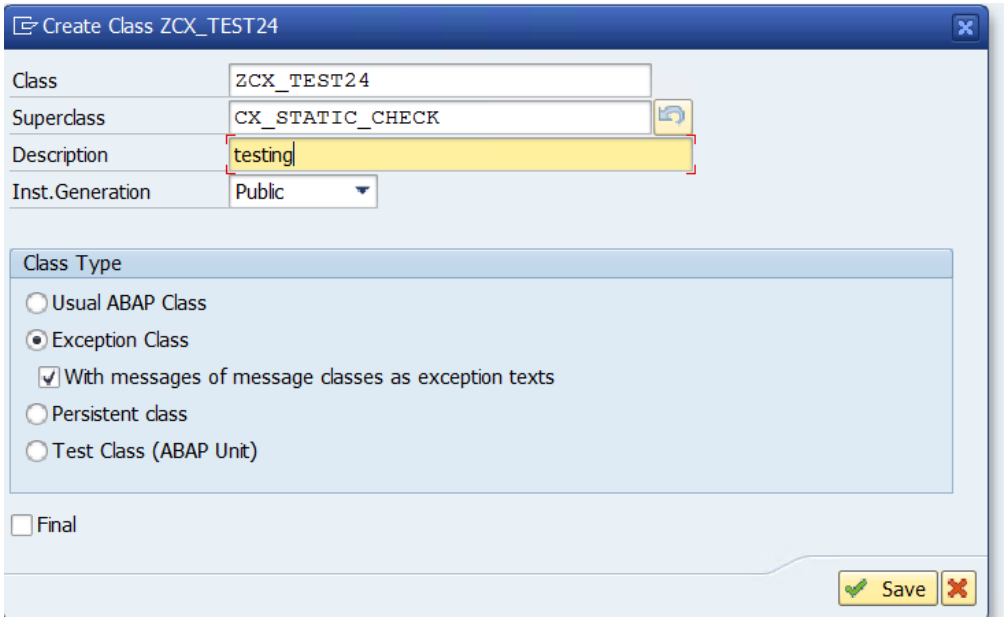
```
DATA obj_ex2 TYPE REF TO cx_sy_conversion_no_number.
PARAMETERS: p_a TYPE string,
            p_b TYPE string.

START-OF-SELECTION.
  CREATE OBJECT obj.

  TRY.
      CALL METHOD obj->m1
        EXPORTING
          i_a = p_a
          i_b = p_b
        IMPORTING
          i_c = DATA(lv_c).
    CATCH cx_sy_zerodivide INTO obj_ex.
      CALL METHOD obj_ex->get_longtext
        RECEIVING
          result = DATA(lv_result1).

      WRITE:/ 'By using get_longtext method message -', lv_result1.

    CATCH cx_sy_conversion_no_number INTO obj_ex2.
      CALL METHOD obj_ex2->get_longtext
        RECEIVING
          result = DATA(lv_result2).

      WRITE:/ lv_result2.
  ENDTRY.
```

Execute the program by giving p_b = 'ABC', output will come like this.

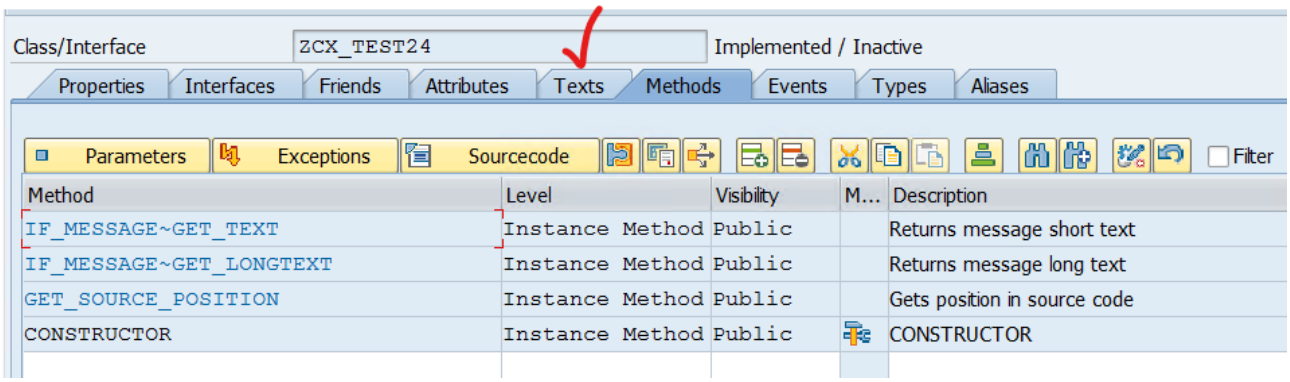
```
testing
---------------------------------------------------------------------
The argument 'ABC' cannot be correctly displayed as a number.
```

In one single Try and End try block we can maintain any number of catch blocks.

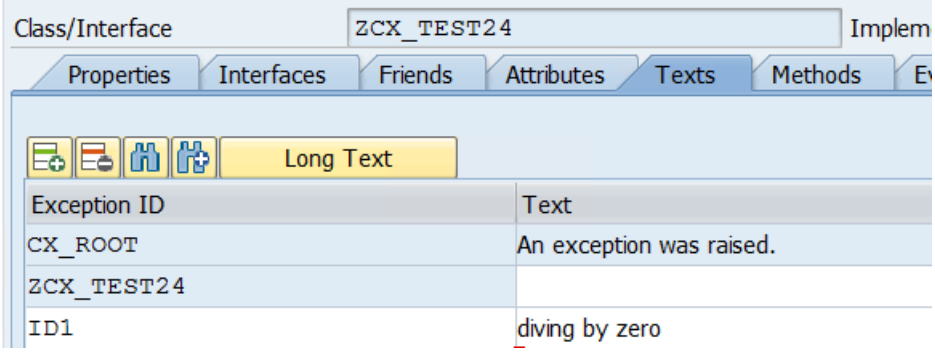
We can handle exceptions by using custom exception class also.
Create one custom class. The class name should start with 'ZCX' or 'YCX'.



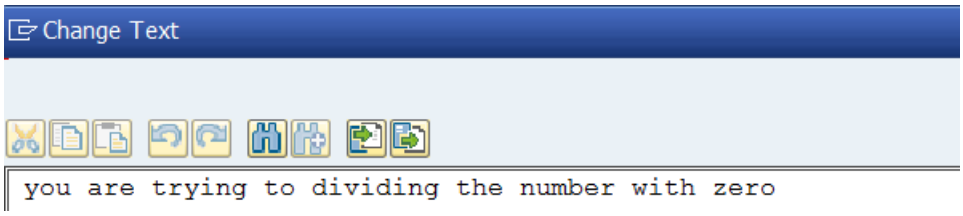
I've given exception class name as ZCX_TEST24, clicked on 'Create' button, automatically one super class name is coming like CX_STATIC_CHECK, we can change it to CX_DYNAMIC_CHECK also. We need to maintain any one super class name for exception class. By default the second radio button – Exception class is selecting. As of now de select the checkbox – 'With message of message classes as exception texts', and click on 'SAVE' button.



By default methods are coming from super class and we can see on extra tab named as 'Texts'. We need to provide text ids here with message. Click on 'Texts' tab provide the text ids.

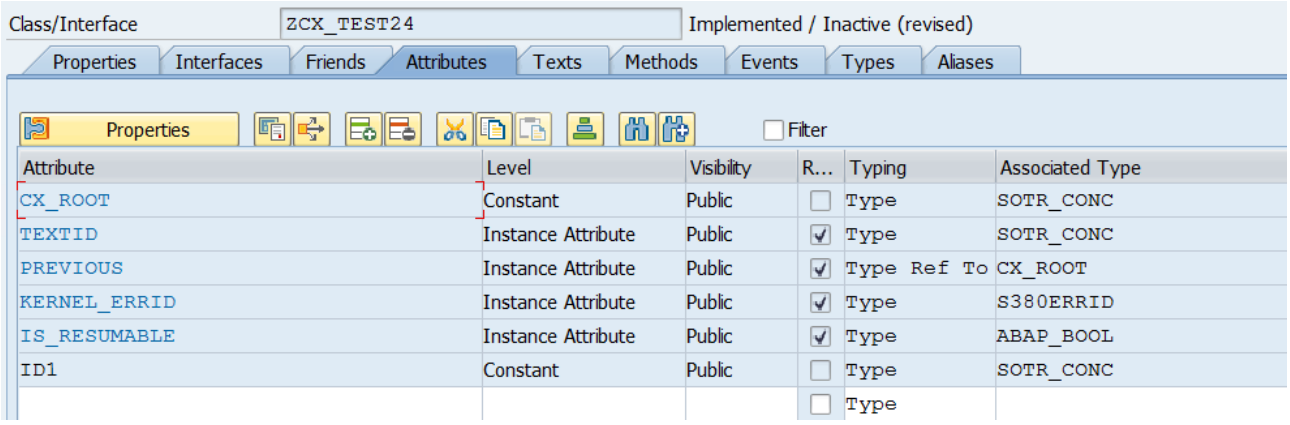


By default CX_ROOT, ZCX_TEST24 are coming. I've provided id1 and message (GET_TEXT will handle it). If we want to provide long test, click on text id, and click on 'Long Text' button, provide the long text (GET_LONGTEXT method will take about it).



Click on ok, save button.

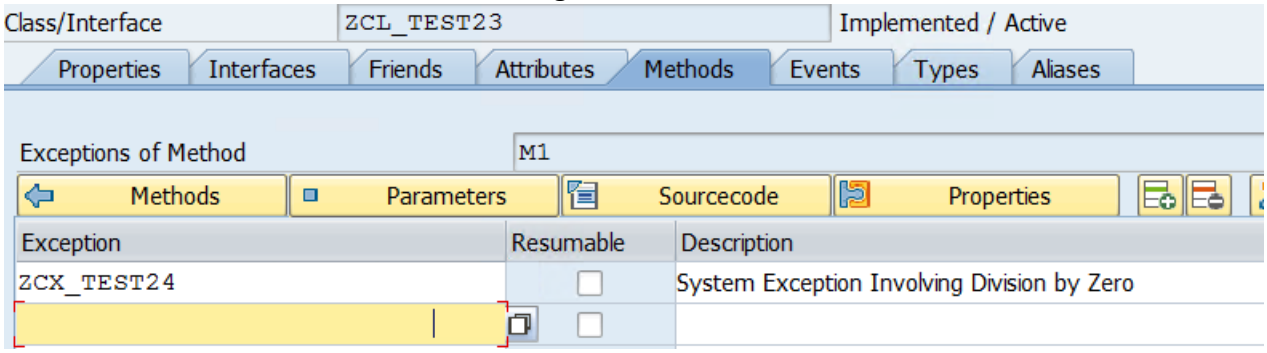
Click on 'Attributes' tab.



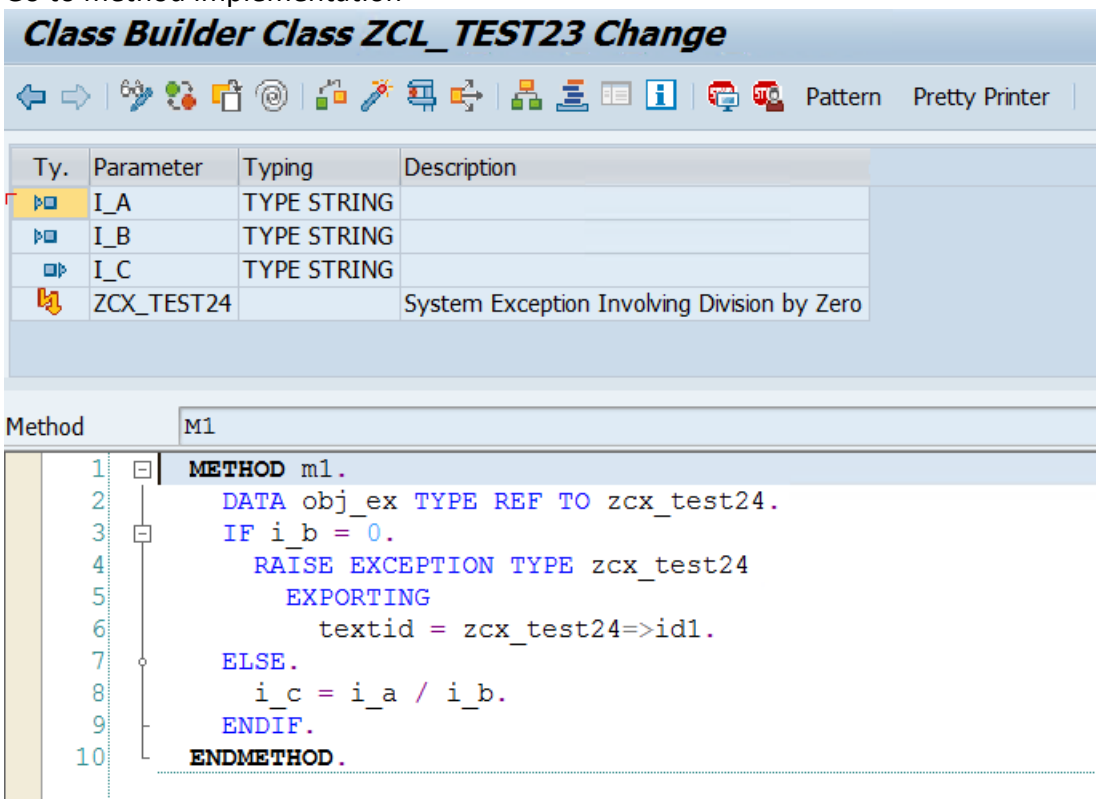
We can see a few default attributes which are coming from super class. I've added one text id or exception id in 'Texts' tab, that will come to attributes tab.

Save and activate the class.

Now come to the class where we are diving the variables.



Go to method implementation



If input value is '0', then we are raising the exception. If we are using standard exception class, then not required this raising part. Automatically it will raise. If we use custom exception class, then we need to raise the exception externally.

We can handle the exception in method calling.

```
DATA obj TYPE REF TO zcl_test23.
DATA obj_ex TYPE REF TO zcx_test24.
PARAMETERS: p_a TYPE string,
            p_b TYPE string.

START-OF-SELECTION.
  CREATE OBJECT obj.

  TRY.
      CALL METHOD obj->m1
```

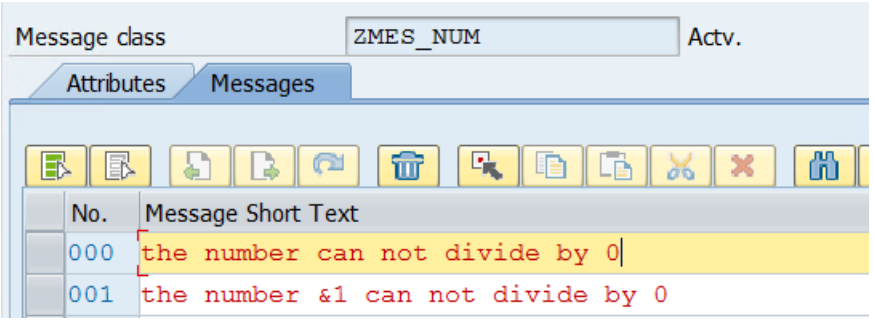
```
        EXPORTING
          i_a = p_a
          i_b = p_b
        IMPORTING
          i_c = DATA(lv_c).
    CATCH zcx_test24 INTO obj_ex.
      CALL METHOD obj_ex->get_longtext
        RECEIVING
          result = DATA(lv_result1).
      WRITE:/ 'By using get_longtext method message -', lv_result1.
  ENDTRY.
```

Execute the program by giving input p_b = 0.

```
testing
By using get_longtext method message - you are trying to dividing the number with zero
```

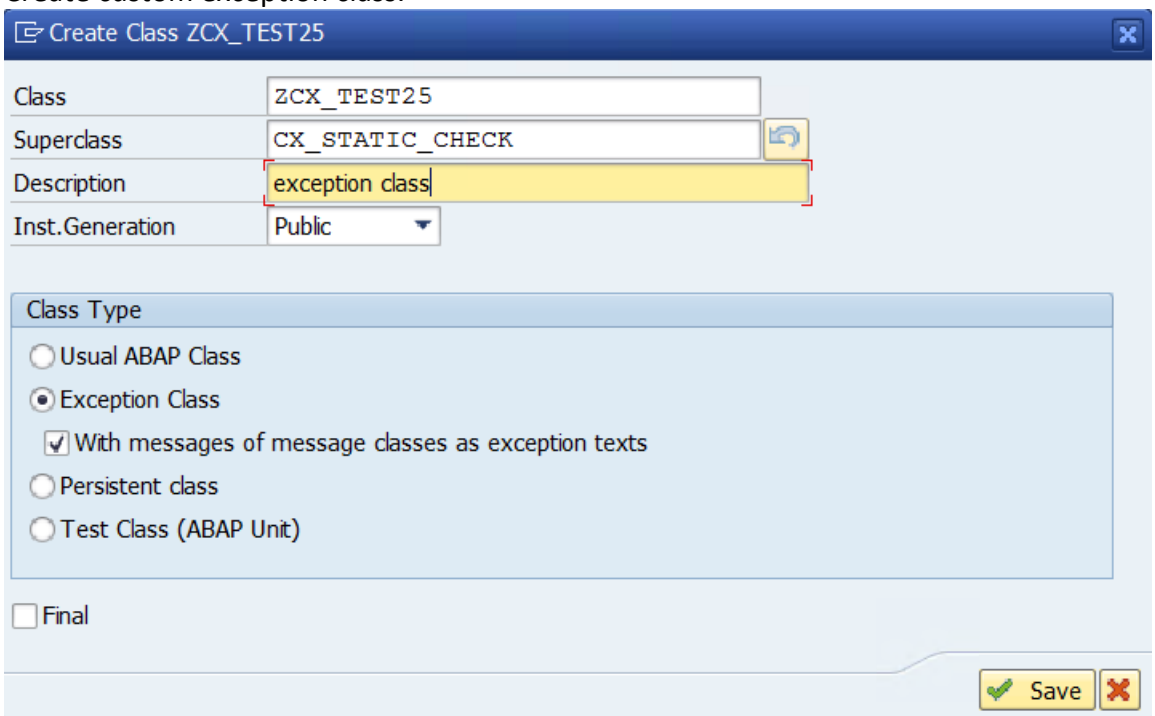
If we want to display messages dynamically, then we need to maintain the messages in Global message class (SE91 transaction). We can assign those messages in exception class. That exception class will use in our normal class.

Go to SE91 transaction.



Provide the message like this. In 001, I've mentioned like &1. We can add dynamic value in place of &1.

Create custom exception class.



Select the check box under Exception Class radio button. If we select the check box then only we can add global message class texts in exception class.

Class/Interface ZCX_TEST25 Implemented / Active

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

Properties | Filter

Attribute	Level	Visibility	R...	Typing	Associated Type
IF_T100_MESSAGE~DEFAULT_TEXTID	Constant	Public	☐		
IF_T100_MESSAGE~T100KEY	Instance Attribute	Public	☐	Type	SCX_T100KEY
IF_T100_DYN_MSG~MSGV1	Instance Attribute	Public	☐	Type	SYMSGV
IF_T100_DYN_MSG~MSGV2	Instance Attribute	Public	☐	Type	SYMSGV
IF_T100_DYN_MSG~MSGV3	Instance Attribute	Public	☐	Type	SYMSGV
IF_T100_DYN_MSG~MSGV4	Instance Attribute	Public	☐	Type	SYMSGV
IF_T100_DYN_MSG~MSGTY	Instance Attribute	Public	☐	Type	SYMSGTY
CX_ROOT	Constant	Public	☐	Type	SOTR_CONC
TEXTID	Instance Attribute	Public	☑	Type	SOTR_CONC
PREVIOUS	Instance Attribute	Public	☑	Type Ref To	CX_ROOT
KERNEL_ERRID	Instance Attribute	Public	☑	Type	S380ERRID
IS_RESUMABLE	Instance Attribute	Public	☑	Type	ABAP_BOOL
ID1	Constant	Public	☐		
ID2	Constant	Public	☐		
LV_TEXT1	Static Attribute	Public	☐	Type	STRING

Add one Static attribute to hold the dynamic value.

Click on 'Texts' tab.

Class/Interface ZCX_TEST25 Implemented / Inactive

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

Long Text | Message Text

Exception ID	Text
ZCX_TEST25	
id1	
id2	

We can give exception id but text is disable mode. Click on exception id, click on 'Message Text' button.

Assign Attributes of an Exception Class to a Message

Message Class ZMES_NUM

Message Number 000

Message Text

Attributes for Exception Class

Attrib. 1

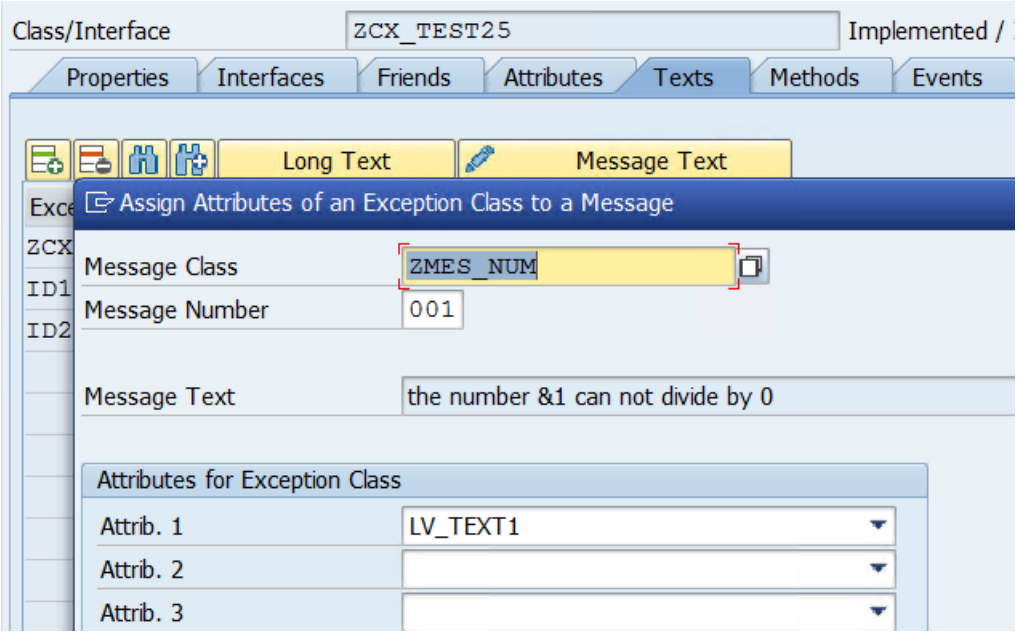
Attrib. 2

Attrib. 3

Attrib. 4

Change

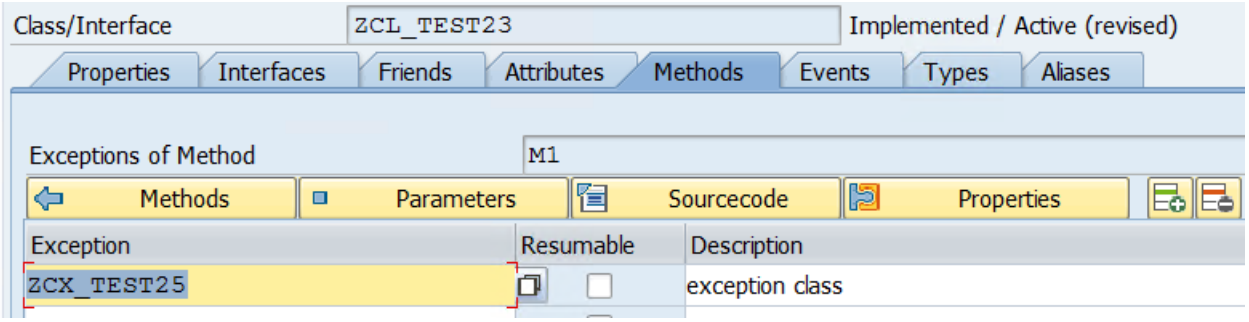
Select the second id, click on 'Message Text' button.



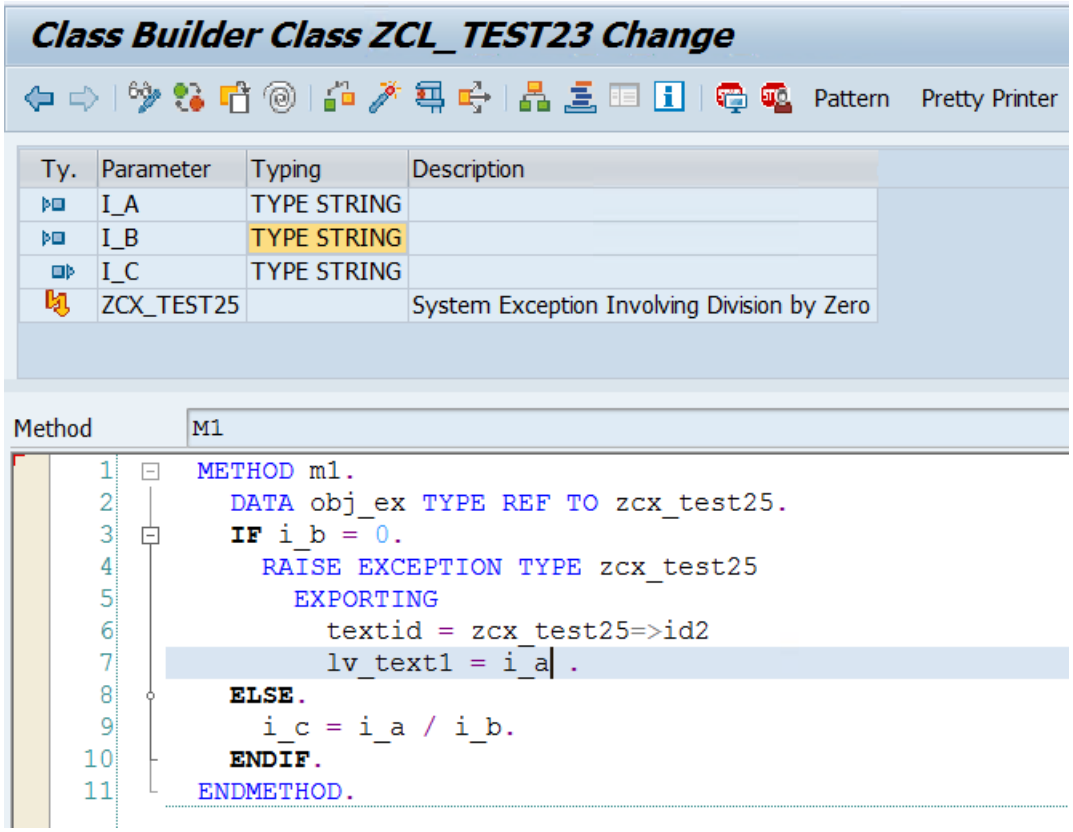
Provide message class, message number and in attribute 1, provide static variable name.

Save, check, activate the exception class.

Go to normal class which is raising the exception class.



Go to method implementation



Go to the program.

```
REPORT ztest_28731.
DATA obj TYPE REF TO zcl_test23.
DATA obj_ex TYPE REF TO zcx_test25.
PARAMETERS: p_a TYPE string,
            p_b TYPE string.

START-OF-SELECTION.
  CREATE OBJECT obj.

  TRY.
      CALL METHOD obj->m1
        EXPORTING
          i_a = p_a
          i_b = p_b
        IMPORTING
          i_c = DATA(lv_c).
    CATCH zcx_test25 INTO obj_ex.
      CALL METHOD obj_ex->get_longtext
        RECEIVING
          result = DATA(lv_result1).
      WRITE:/ 'By using get_longtext method message -', lv_result1.
  ENDTRY.
```

Execute the program by giving input as ‘0’.

```
testing
-------------------------------------------------------------------------------
By using get_longtext method message - the number 20 can not divide by 0.
```

In the message the number 20 is coming as dynamic. We giving input with different number, the same will apply in message also.

Interface

Interface is the collection methods which are declare and not implemented.

1. It is independent structure, not having method implementation
2. It has been used to extend the functionality of class
3. Reusability and maintain the frame work of the project
4. It can be used in number of classes depending on design

Interface in SE24 transaction:

Interface ZIF_TEST123 Implemented / Active

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

Parameters | Exceptions | Filter

Method	Level	M...	Description
SALES_INFO	Instance Method		Sales Information
DEL_INFO	Instance Method		Delivery Information

By default all methods will be under Public section. So system will not show visibility section here. There is not 'Source Code' button because we can not implement the method in Interface.

Create one more interface

Interface ZIF_TEST124 Implemented / Active

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

Parameters | Exceptions | Filter

Method	Description
PR_INFO	Purchase Requisition Information
PO_INFO	Purchase Order Information

Use these two interfaces in one class.

Class/Interface ZCL_TEST_INTF1 Implemented / Active

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

Properties | Filter

Interface	Abstract	Final	Modeled...	Description
ZIF_TEST123				testing
ZIF_TEST124				TESTING

In interfaces tab, need to add the interface names. If we click on Methods tab, it will show list of methods from interfaces.

ZIF_TEST124~PR_INFO	Instance Method	Public		Purchase Requisition Information
ZIF_TEST124~PO_INFO	Instance Method	Public		Purchase Order Information
ZIF_TEST123~SALES_INFO	Instance Method	Public		Sales Information
ZIF_TEST123~DEL_INFO	Instance Method	Public		Delivery Information
M1	Instance Method	Public		Method M1

By default 2 interfaces methods are coming. Method name will come like 'Interface name ~ method name' with different background colour.
Instead of calling method name by using, interface name ~ method name, we can go for alias name and give any other name for interface methods.

ZIF_TEST123~CUST_NUM		Public	CUST_NUM
ZIF_TEST123~DEL_INFO		Public	DEL_INFO
ZIF_TEST123~SALES_INFO		Public	SALES_INFO
ZIF_TEST124~PO_INFO		Public	PO_INFO
ZIF_TEST124~PR_INFO		Public	PR_INFO

Implemented the methods like this.

```
method ZIF_TEST124~PR_INFO.
  write:/ 'PR Infro method calling from ZCL_TEST_INTF1'.
endmethod.

method ZIF_TEST124~PO_INFO.
  WRITE:/ 'PO INFO method calling from ZCL_TEST_INTF1'.
endmethod.

method ZIF_TEST123~SALES_INFO.
  write:/ 'Method SALES_INFO calling from ZCL_TEST_INTF1'.
endmethod.

method ZIF_TEST123~DEL_INFO.
  WRITE:/ 'Method DEL_INFO from ZCL_TEST_INTF1'.
endmethod.

METHOD m1.
  zif_test123~cust_num = '123'.
  WRITE:/ 'Method M1 is passing value to custmer as ', zif_test123~cust_num.
ENDMETHOD.
```

Calling the methods from local program.

```
REPORT ztest_28731.
DATA obj TYPE REF TO ZCL_TEST_INTF1.
START-OF-SELECTION.
create object obj.

call method obj->m1.
call method obj->sales_info.
call method obj->del_info.
call method obj->po_info.
call method obj->pr_info.
```

```
testing
---------------------------------------------------------
Method M1 is passing value to custmer as  123
Method SALES_INFO calling from ZCL_TEST_INTF1
Method DEL_INFO from ZCL_TEST_INTF1
PO INFO method calling from ZCL_TEST_INTF1
PR Infro method calling from ZCL_TEST_INTF1
```

Note: Multiple inheritance is not possible directly. By using interfaces, it is possible.

Abstract Class

1. Abstract method can be defined in abstract class
2. If normal class contain abstract method, at the time of activation automatically the class will convert as Abstract class
3. Implementation is possible for Abstract class
4. Abstract method Implementation should be in sub class
5. We can't create object for Abstract class
6. Through sub class object, we can access the Abstract class properties
7. Abstract class can contain Instance, Static, Abstract methods.

Differences between Abstract class and Interface

	Abstract Class	Interface
1	Multiple Inheritance is not possible	Multiple Inheritance is possible
2	Partial implementation in this class	Implementation is not possible
3	Method can be in Public or Protected or Private	By default all methods will come under Public
4	It can contain both Normal method and Abstract Methods	It should contain only Abstract methods
5	Explicitly need to declare as abstract methods	By default method should be abstract

Persistence Class

Persistence Service: it is the layer in between of ABAP program and data base table which enables to store the attributes of the object with unique identity and when ever requires it retrieves from the data base in the same state as it was at the time of saving.

Persistence Class: - A class which enables the Persistence Service is known as Persistence Class.

If we create something class and object, Object information will clear once we close the program. If we want save the last update information (Object), then we have to go for Persistence Class. To save the last update Object information, we need to create one table. Here, I've created a table.

Transparent Table: ZSALES_ORD_INFO — Active

Short Description: SALES ORDER INFORMATION

Attributes | Delivery and Maintenance | Fields | Input Help/Check | Currency/Quantity Fields | Indexes

Search | Built-In Type

Field	Key	Initi...	Data element	Data Type	Length	Decim...	Coordinate	Short Description
MANDT	☑	☑	MANDT	CLNT	3	0	0	Client
VBELN	☑	☑	VBELN_VA	CHAR	10	0	0	Sales Document
KUNNR	☐	☐	KUNNR	CHAR	10	0	0	Customer Number
VBTYP	☐	☐	VBTYP	CHAR	1	0	0	SD document category

Create the Persistence class.

Execute SE24. Provide the class name (ZCL_SO_INFO).

Create Class ZCL_SO_INFO

Class: ZCL_SO_INFO

Description: SO INFORMATION

Inst.Generation: Protected

Class Type

- ○ Usual ABAP Class
- ○ Exception Class
  - ☑ With messages of message classes as exception texts
- ◉ Persistent class
- ○ Test Class (ABAP Unit)

☐ Final

Save

Once we select the radio button – Persistent class, automatically the Inst. Generation will change from 'PUBLIC' to 'Protected'.

System automatically add one interface with its methods.

Class/Interface: ZCL_SO_INFO — Implemented / Inactive

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

Properties | Filter

Interface	Abstract	Final	Modeled...	Description
IF_OS_STATE	☐	☐	☐	State Management for a 'Managed Object'

If we open the methods tab,

Class/Interface ZCL_SO_INFO Implemented / Inactive

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

Parameters | Exceptions | Sourcecode | Filter

Method	Level	Visibility	M...	Description
IF_OS_STATE~HANDLE_EXCEPTION	Instance Method	Public		Handles Exception After Reading State
IF_OS_STATE~GET	Instance Method	Public		Object Services Private: Copy State Object
IF_OS_STATE~INIT	Instance Method	Public		Initializes Transient Part of Object State
IF_OS_STATE~SET	Instance Method	Public		Object Services Private: Replace State Object
IF_OS_STATE~INVALIDATE	Instance Method	Public		Invalidate Object State

Now we need to attach the table information to the class.
Click on Go to (menu button) → Persistent Representant → Provide the table name - ZSALES_ORD_INFO, click on enter button.

Change Persistence Representation: ZCL_SO_INFO

Generator Settings | Insert Table/Structure

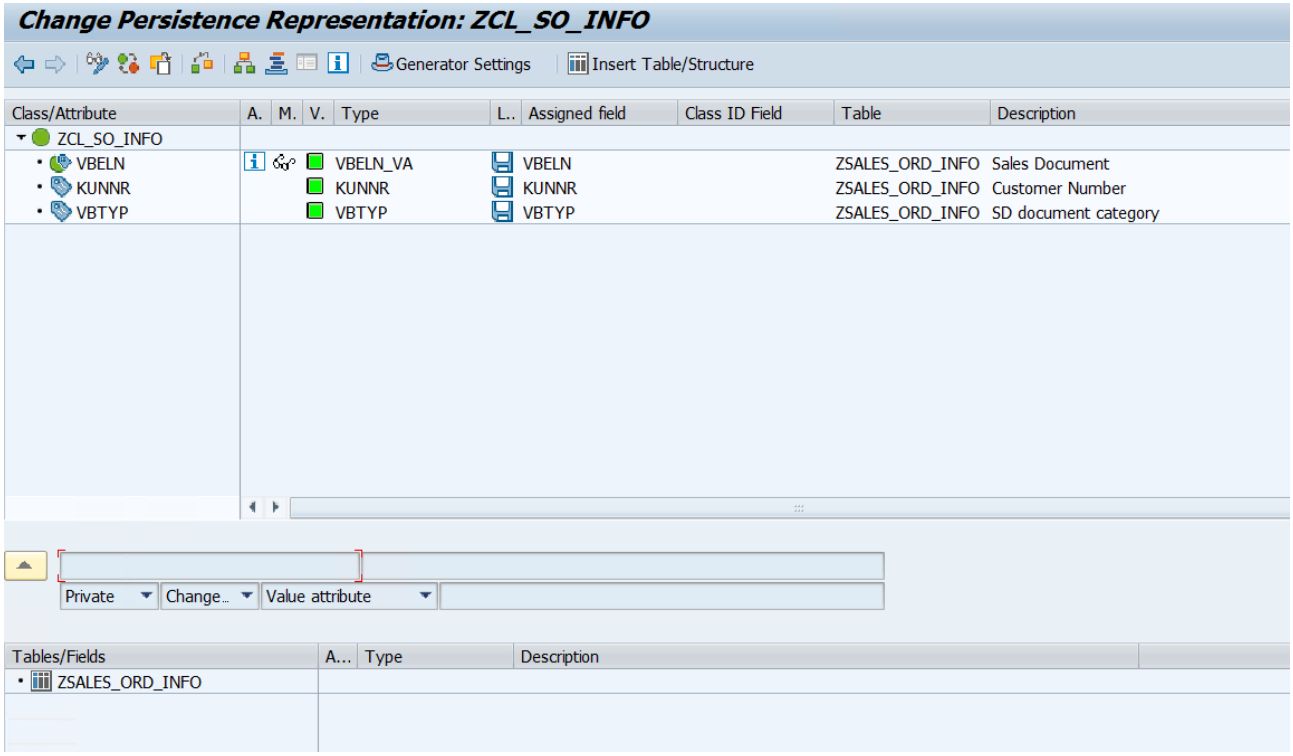
Class/Attribute	A.	M.	V.	Type	L..	Assigned field	Class ID Field	Table
ZCL_SO_INFO								

VBELN Sales Document

Public | Read on... | Business key | VBELN_VA

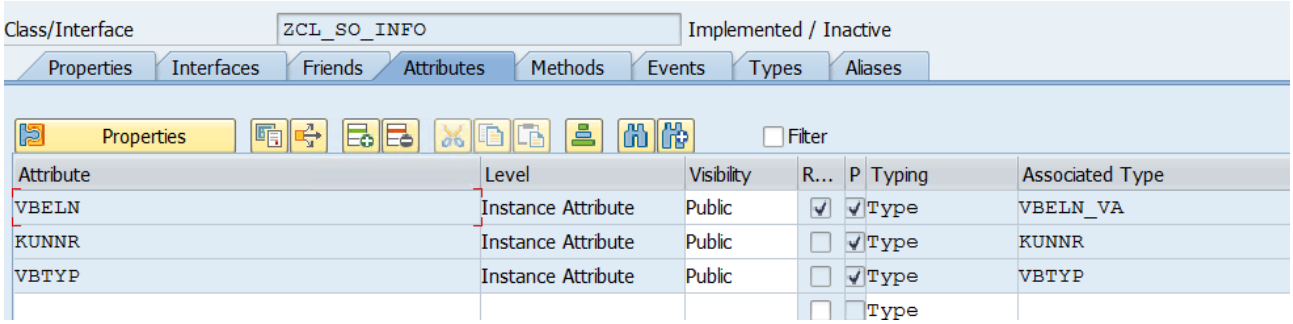
Tables/Fields	A...	Type	Description
ZSALES_ORD_INFO			
VBELN		VBELN_VA	Sales Document
KUNNR		KUNNR	Customer Number
VBTYP		VBTYP	SD document category

Double click on the field VBELN (last window), the field will come to bit top beside to the up arrow mark. Click on that up arrow mark. The field will add in main area. Do the same thing for all fields.



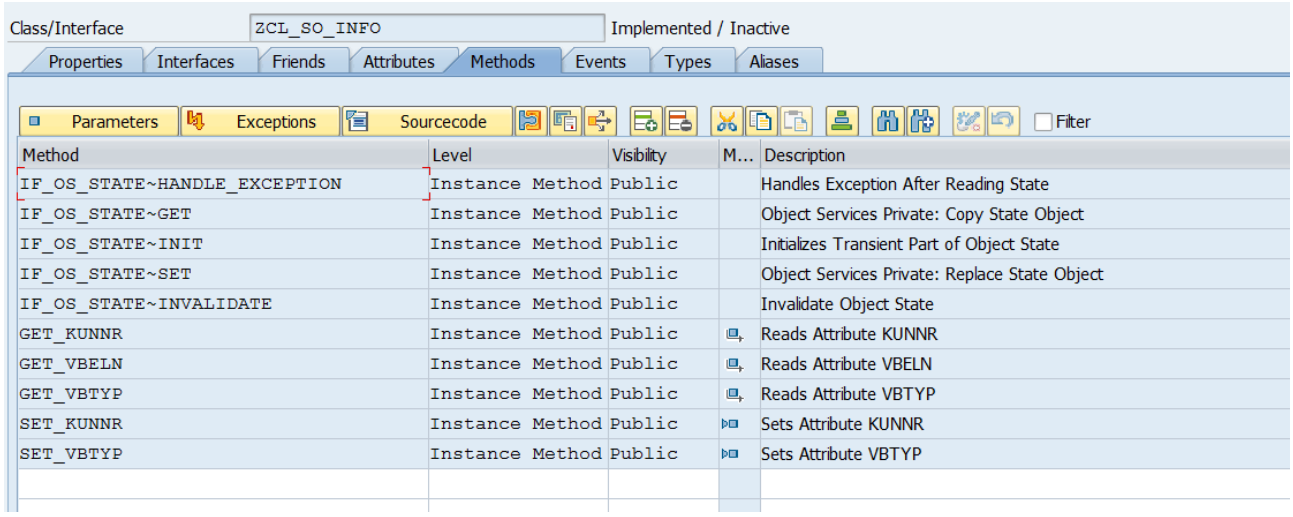
Click on save button.

System will add a few attributes and methods.



In table how many fields are there, that many of attributes it will add here.

If we go for Method tab,



System is adding a few methods like GET_* and SET_*.

In table, for all fields system will create GET_* method and for non key fields also system add methods like SET_*.

System will create one Friend class for this Persistent class.

Class/Interface ZCL_SO_INFO Implemented / Inactive

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

Friend	Mo...	Description
ZCB_SO_INFO		Base agent SO INFORMATION

We will call this as Base Agent Class.
If we open this class, this is Abstract method. So we can't create object for it.

Class/Interface ZCB_SO_INFO Implemented / Inactive

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

Superclass | Undo Inheritance | Change Inheritance

Superclass: CL_OS_CA_COMMON — Modeled Only

Description: Base agent SO INFORMATION

Instance Generation: Abstract

Final

Base agent class will add a few methods like this.
If we activate the Persistent Class then system will add a few methods in Base Agent class.

Class/Interface ZCB_SO_INFO Implemented / Active

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

Parameters | Exceptions | Sourcecode | Filter

Method	Level	Visibility	M...	Description
CREATE_PERSISTENT	Instance Method	Public		Generate New Persistent Object with Key
CREATE_TRANSIENT	Instance Method	Public		Generate Transient Object with Key
DELETE_PERSISTENT	Instance Method	Public		Deletes Persistent Object with Key
GET_PERSISTENT	Instance Method	Public		Get Persistent Object with Key
GET_TRANSIENT	Instance Method	Public		Get Transient Object with Key
MAP_EXTRACT_IDENTIFIER	Instance Method	Protected		Extracts BKey from Database Data
MAP_GET_ATTRIBUTES	Instance Method	Protected		Reads Persistent Attributes from Object

CREATE_PERSISTENT, DELETE_PERSISTENT, GET_PERSISTENT…

System will create one more class at the time of activating the Persistent class. We will call it as Actor class.

Class/Interface ZCA_SO_INFO Implemented / Inactive

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

Superclass | Undo Inheritance | Change Inheritance

Superclass: ZCB_SO_INFO — Modeled Only

Description: Agent SO INFORMATION

Instance Generation: Private

This is sub class for Base class – ZCB_SO_INFO and it is Private class, so we can't access the properties of this class directly.
The naming convention for the classes
ZCL → Persistence Class
ZCB → Base Agent Class
ZCA → Actor Class

Class/Interface ZCA_SO_INFO Implemented / Active

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

Properties | Filter

Attribute	Level	Visibility	R...	Typing	Associated Type	Description
SPECIAL_OBJECT_INFO	Instance Attribute	Protected	☐	Type	TYP_SPECIAL_OBJECT_INFO_TAB	Class-Specific Object Data
AGENT	Static Attribute	Public	☑	Type Ref To	ZCA_SO_INFO	Class Agent - Singleton
			☐	Type		

We can see one static attribute in Actor class. The association is like AGENT TYPE REF TO ZCA_SO_INFO.

Means, it is referring one object for the same class.

If we access this attribute by using class name, if any constructor is there, it will execute automatically. Open the methods tab and check for constructor.

Class/Interface ZCA_SO_INFO Implemented / Active

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

Parameters | Exceptions | Sourcecode | Filter

Method	Level	Visibility	M...	Description
CLASS_CONSTRUCTOR	Static Method	Public		Class Constructor

One Static Constructor is there. Open the method implementation.

Method CLASS_CONSTRUCTOR active

```
23
24    create object AGENT.
25
26    call method AGENT->REGISTER_CLASS_AGENT
27      exporting
28        I_CLASS_NAME          = 'ZCL_SO_INFO'
29        I_CLASS_AGENT_NAME    = 'ZCA_SO_INFO'
30        I_CLASS_GUID          = '3F2324D4F0921EEDB2C9D6B1E63ACC27'
31        I_CLASS_AGENT_GUID    = '3F2324D4F0921EEDB2C9D6B1E63B4C27'
32        I_AGENT               = AGENT
33        I_STORAGE_LOCATION    = 'ZSALES_ORD_INFO'
34        I_CLASS_AGENT_VERSION = '2.0'.
35
36              "CLASS_CONSTRUCTOR
37    endmethod.
```

This constructor will execute by default, and it will provide one object for this class.

Create one local program.

```
REPORT ztest_persistence.
PARAMETERS p_vbeln TYPE vbeln_va.
PARAMETERS p_kunnr TYPE kunnr.
PARAMETERS p_vbtyp TYPE vbtyp.
PARAMETERS: p_cr RADIOBUTTON GROUP s USER-COMMAND uc DEFAULT 'X',
            p_dl RADIOBUTTON GROUP s,
            p_gt RADIOBUTTON GROUP s,
            p_ch RADIOBUTTON GROUP s.
DATA obj_actor TYPE REF TO zca_so_info.
DATA obj_per TYPE REF TO zcl_so_info.

START-OF-SELECTION.
  obj_actor = zca_so_info=>agent.   "Object will come here

  IF p_cr = 'X'.
    TRY.
        CALL METHOD obj_actor->create_persistent
          EXPORTING
            i_vbeln = p_vbeln                    " Business Key
```

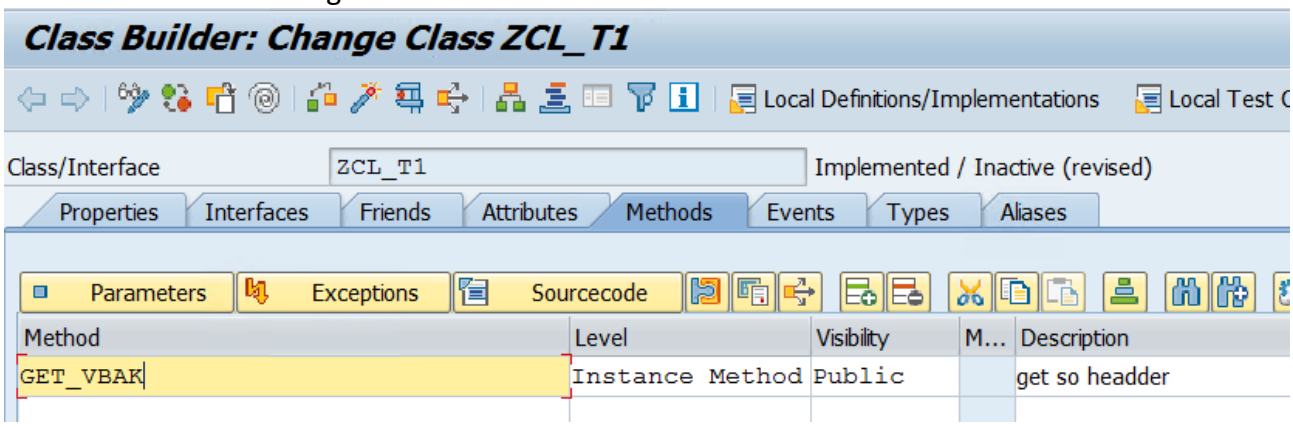
```
          RECEIVING
            result  = obj_per.                  " Newly Generated Persistent
Object
        TRY.
            CALL METHOD obj_per->set_kunnr
              EXPORTING
                i_kunnr = p_kunnr.                  " Attribute Value
            TRY.
                CALL METHOD obj_per->set_vbtyp
                  EXPORTING
                    i_vbtyp = p_vbtyp.                 " Attribute Value
              CATCH cx_os_object_not_found. " Object Services Exception
            ENDTRY.
            COMMIT WORK.
          CATCH cx_os_object_not_found. " Object Services Exception
        ENDTRY.
      CATCH cx_os_object_existing. " Object Services Exception
    ENDTRY.
  ELSEIF p_dl = 'X'.
    p_vbeln = |{ p_vbeln ALPHA = IN }|.
    TRY.
        CALL METHOD obj_actor->delete_persistent
          EXPORTING
            i_vbeln = p_vbeln.                  " Business Key
      CATCH cx_os_object_not_existing. " Object Services Exception
    ENDTRY.
    COMMIT WORK.

  ELSEIF p_GT = 'X'.
    CLEAR: obj_per.
    TRY.
        CALL METHOD obj_actor->get_persistent
          EXPORTING
            i_vbeln = p_vbeln                 " Business Key
          RECEIVING
            result  = obj_per.                " Persistent Object

        IF obj_per IS BOUND.
          CLEAR: p_kunnr, p_vbtyp.
          TRY.
              CALL METHOD obj_per->get_kunnr
                RECEIVING
                  result = p_kunnr.               " Attribute Value
            CATCH cx_os_object_not_found. " Object Services Exception
          ENDTRY.

*           TRY.
*               CALL METHOD obj_per->get_vbtyp
*                 RECEIVING
*                   result = P_VBTYP.               " Attribute Value
*             CATCH cx_os_object_not_found. " Object Services Exception
*           ENDTRY.
        ENDIF.
      CATCH cx_os_object_not_found. " Object Services Exception
    ENDTRY.
  ELSEIF p_ch = 'X'.
    CALL METHOD obj_actor->get_persistent
      EXPORTING
        i_vbeln = p_vbeln                  " Business Key
      RECEIVING
        result  = obj_per.                  " Persistent Object

    IF obj_per IS BOUND.
      CALL METHOD obj_per->set_kunnr
        EXPORTING
```

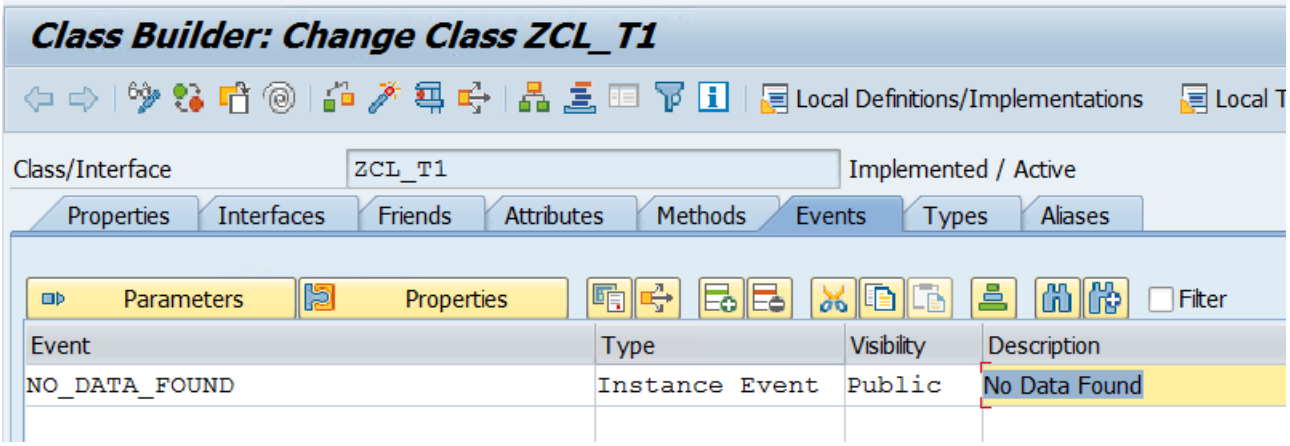
```
          i_kunnr = p_kunnr.                  " Attribute Value
      COMMIT WORK.
      CALL METHOD obj_per->set_vbtyp
        EXPORTING
          i_vbtyp = p_vbtyp.                  " Attribute Value
      COMMIT WORK.
*             CATCH cx_os_object_not_found. " Object Services Exception
    ENDIF.
*       CATCH cx_os_object_not_found. " Object Services Exception
  ENDIF.
```

EVENTS

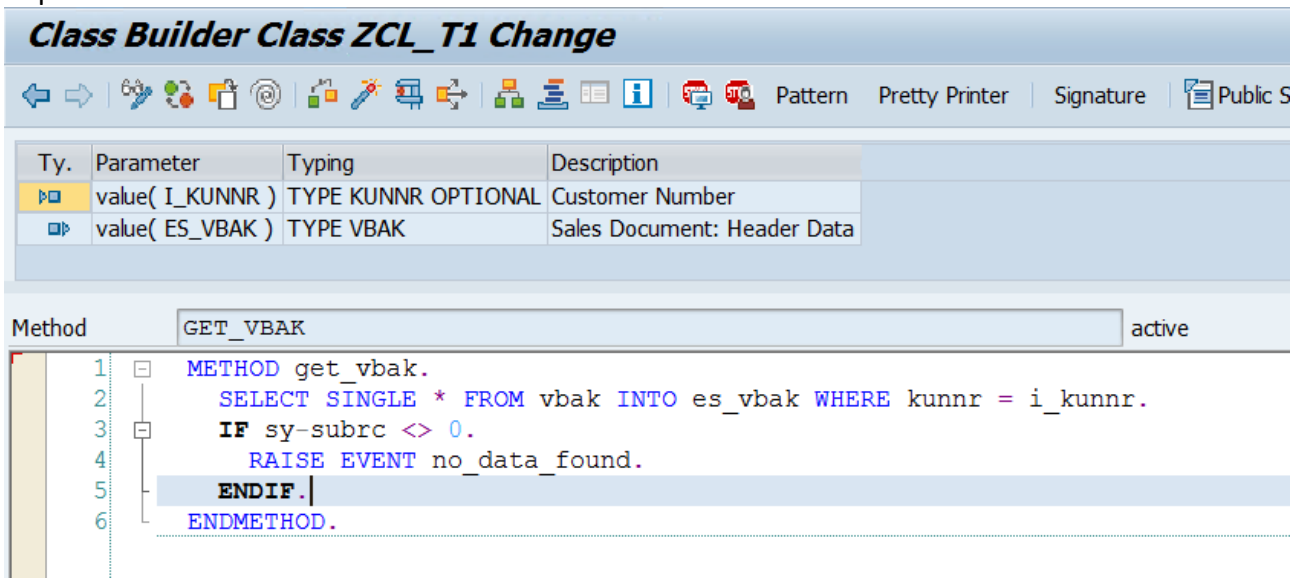
Event is an action to do one particular activity. We can declare events in the class. But Event handle method should in the same class or any other class.

Create a class with adding one event.

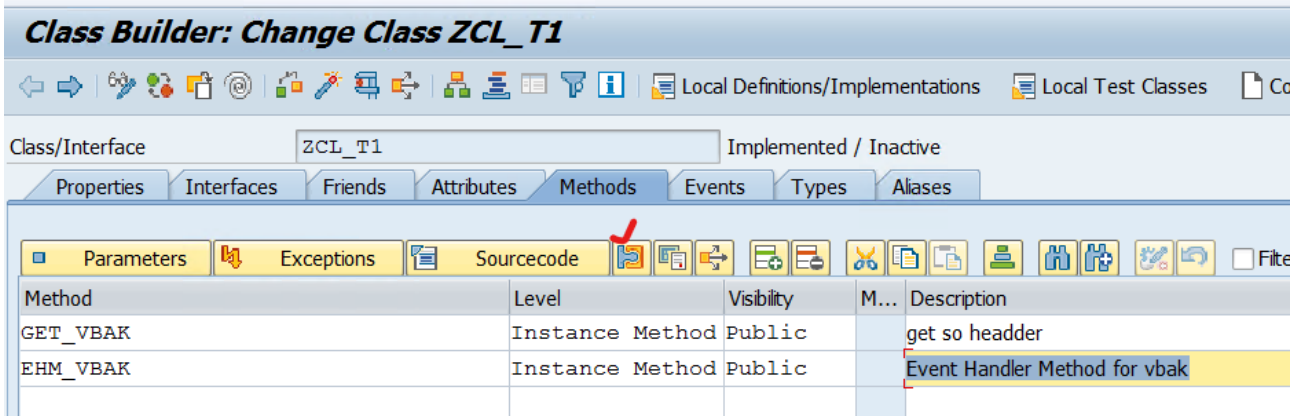




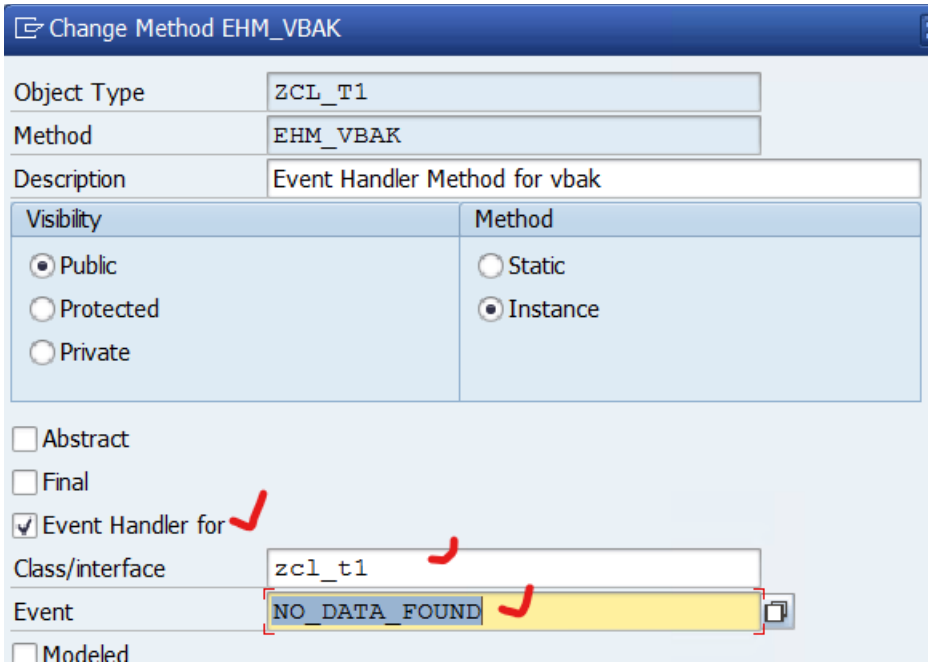
Implement the method.



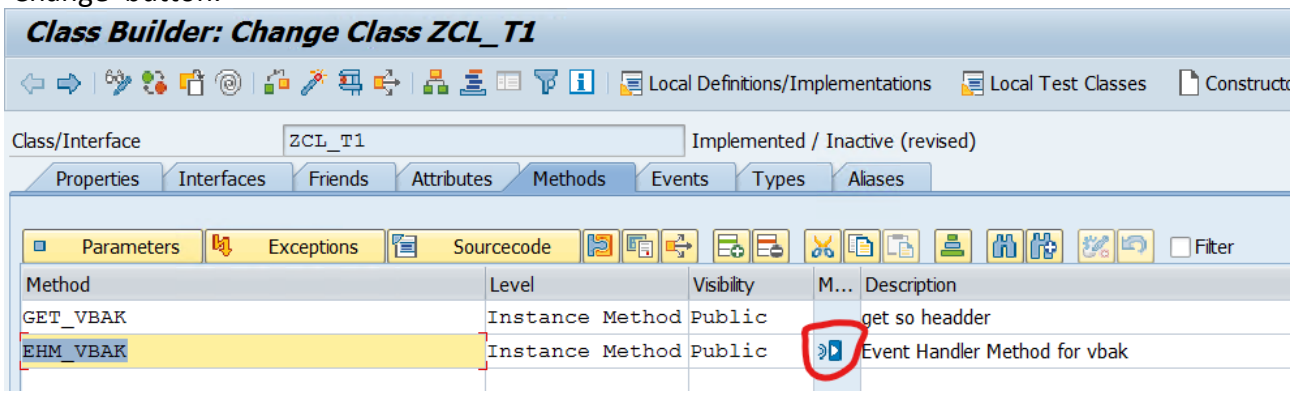
If data is not fetching, then event is raising here. Now need to take one method which will handle this event.



EHM_VBAK is the method which will handle the event. But we need to tell the system that, EHM_VBAK method will handle the event. Select the EHM_VBAK method, click on Go to Parameters.

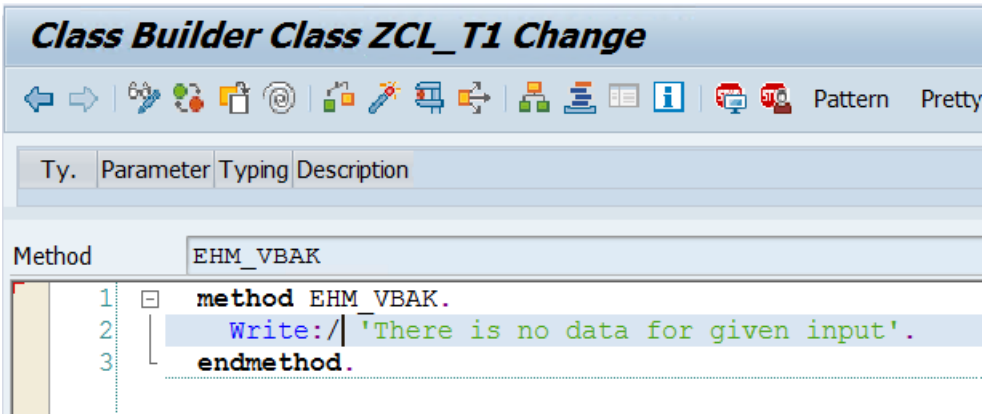


Select the checkbox of 'Event Handler for'. Provide the class name and event name. click on 'Change' button.



Now we can see the symbol of Event Handler Method as shown in the above image.

Implement the Event Handler Method.



Activate the class. Call the method GET_VBAK in one program.

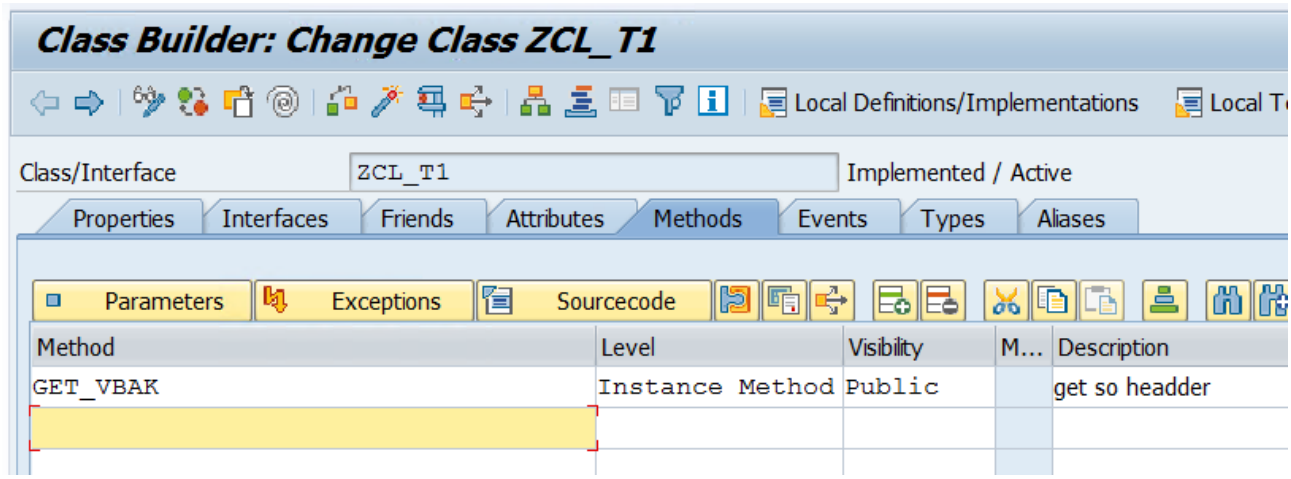
```
REPORT ztest_2931.
DATA obj_t1 TYPE REF TO zcl_t1.
PARAMETERS p_kunnr TYPE kunnr.

START-OF-SELECTION.
  CREATE OBJECT obj_t1.

SET HANDLER obj_t1->ehm_vbak for obj_t1.
  CALL METHOD obj_t1->get_vbak
    EXPORTING
      i_kunnr = p_kunnr                  " Customer Number
    IMPORTING
      es_vbak = DATA(ls_vbak).                   " Sales Document: Header Data

  write:/ ls_vbak-vbeln, ls_vbak-kunnr.
```

Before displaying the output, need to register the event by using below syntax.

```
 SET HANDLER obj_t1->ehm_vbak for obj_t1.
```

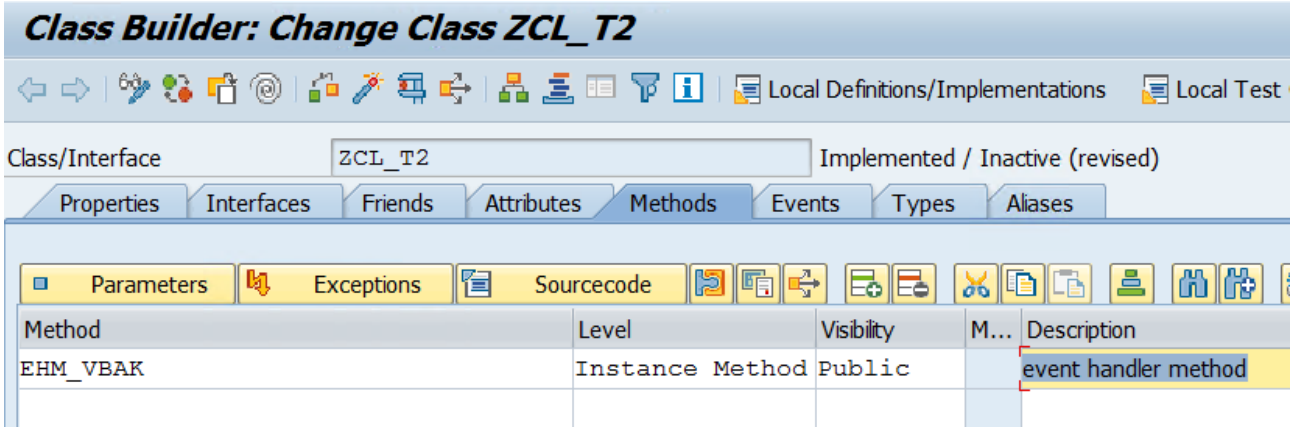
Here, event handler method and normal method both are in the same class. We can maintain the event handler method in another class also.

Remove the event handler method in the present class and create one new class, maintain the event handler method in the new class.

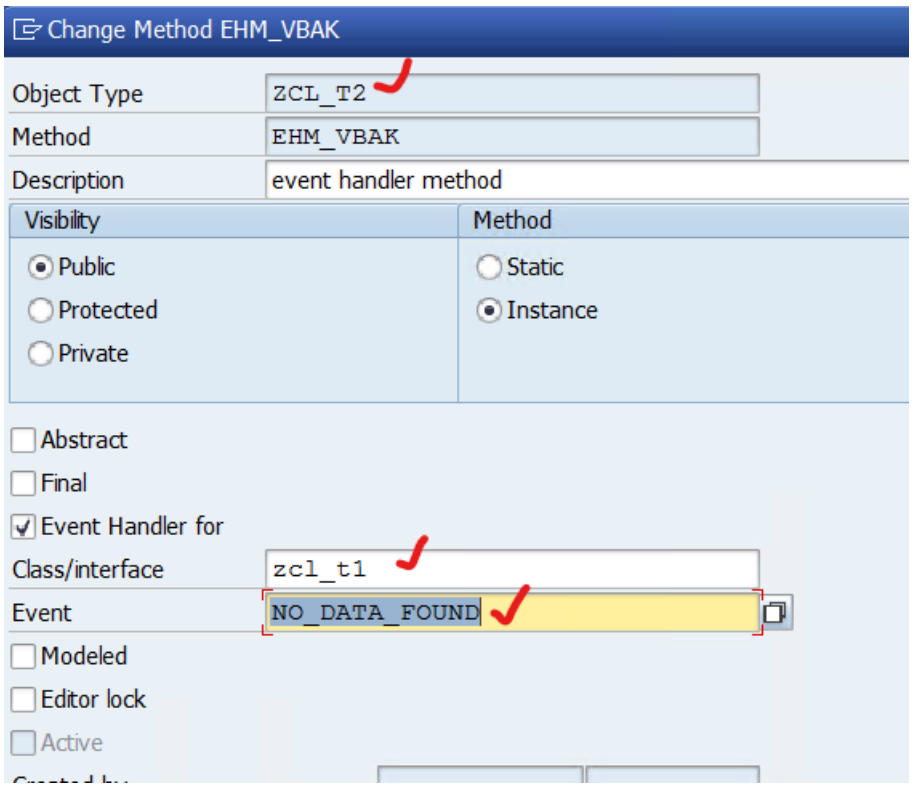


Removed the event handler method from the present class.

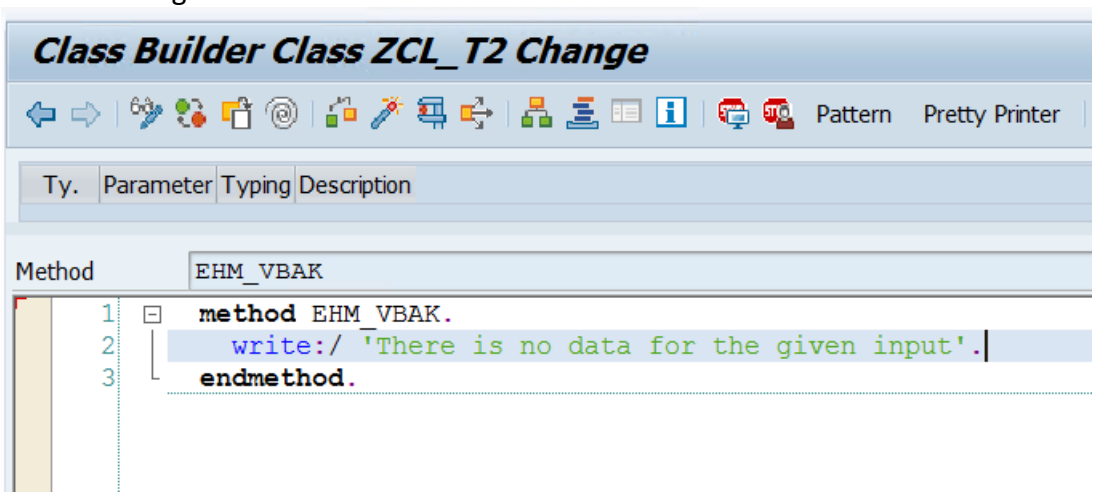
Create one more new class and maintain the event handler method here.



Make the method as event handler method.



Click on change button.

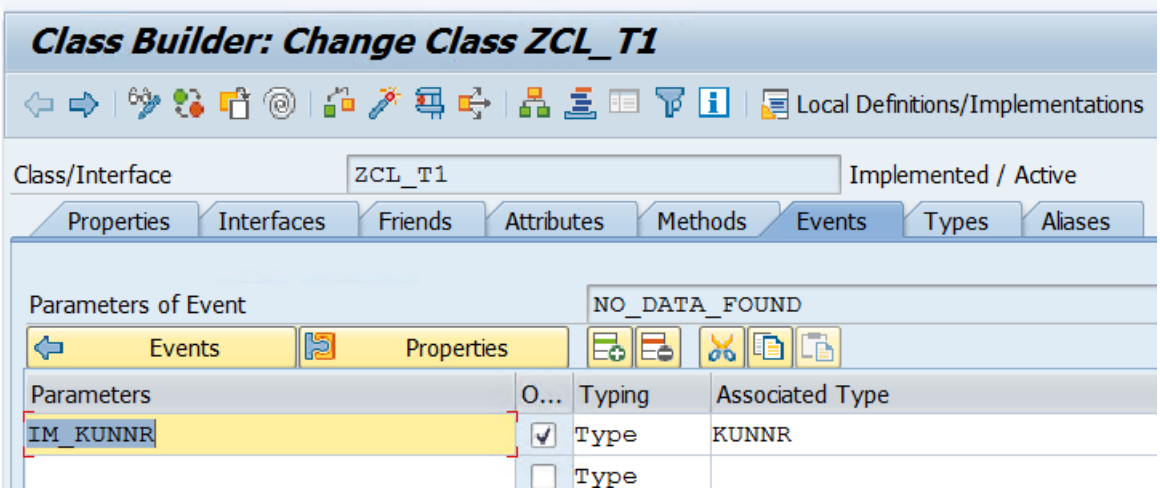


Now go to the program and register the event.

```
REPORT ztest_2931.

DATA obj_t1 TYPE REF TO zcl_t1.
```

```
DATA obj_t2 TYPE REF TO zcl_t2.
PARAMETERS p_kunnr TYPE kunnr.

START-OF-SELECTION.
  CREATE OBJECT obj_t1.
  CREATE OBJECT obj_t2.

set HANDLER obj_t2->ehm_vbak for obj_t1.
  CALL METHOD obj_t1->get_vbak
    EXPORTING
      i_kunnr = p_kunnr                    " Customer Number
    IMPORTING
      es_vbak = DATA(ls_vbak).                     " Sales Document: Header Data

  write:/ ls_vbak-vbeln, ls_vbak-kunnr.
```

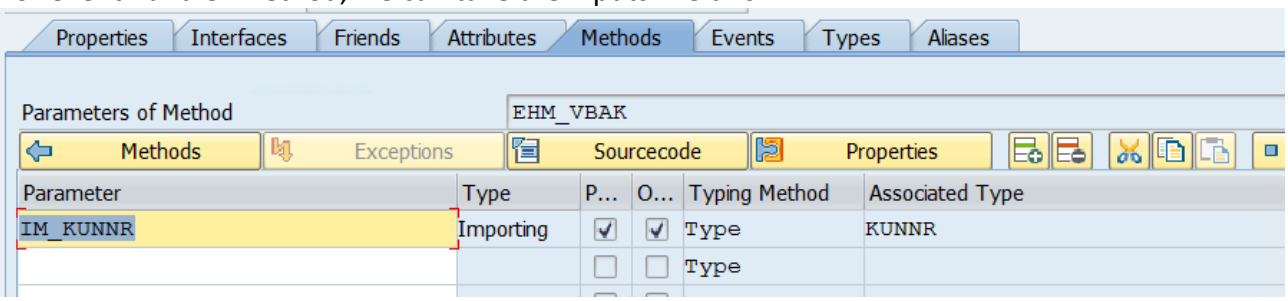
We can give importing parameters for events.



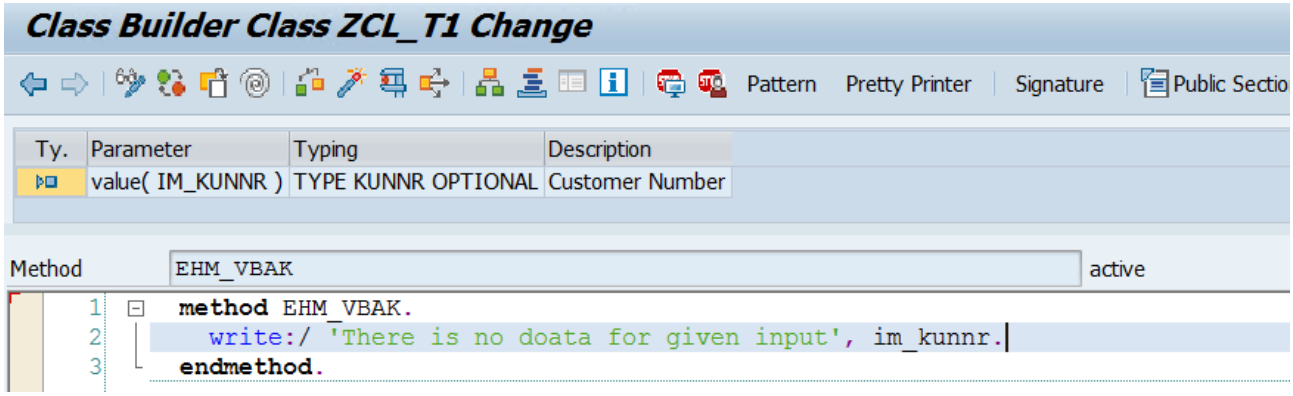
The main method logic should be like this.

```
  METHOD get_vbak.
    SELECT SINGLE * FROM vbak INTO es_vbak WHERE kunnr = i_kunnr.
    IF sy-subrc <> 0.
      RAISE EVENT no_data_found EXPORTING im_kunnr = i_kunnr.
    ENDIF.
  ENDMETHOD.
```

For event handler method, we can take the inputs like this.



We can't give more parameters here. What ever the parameters we have given in event, those parameters should be mention here. If we pass different parameters, system will through errors.



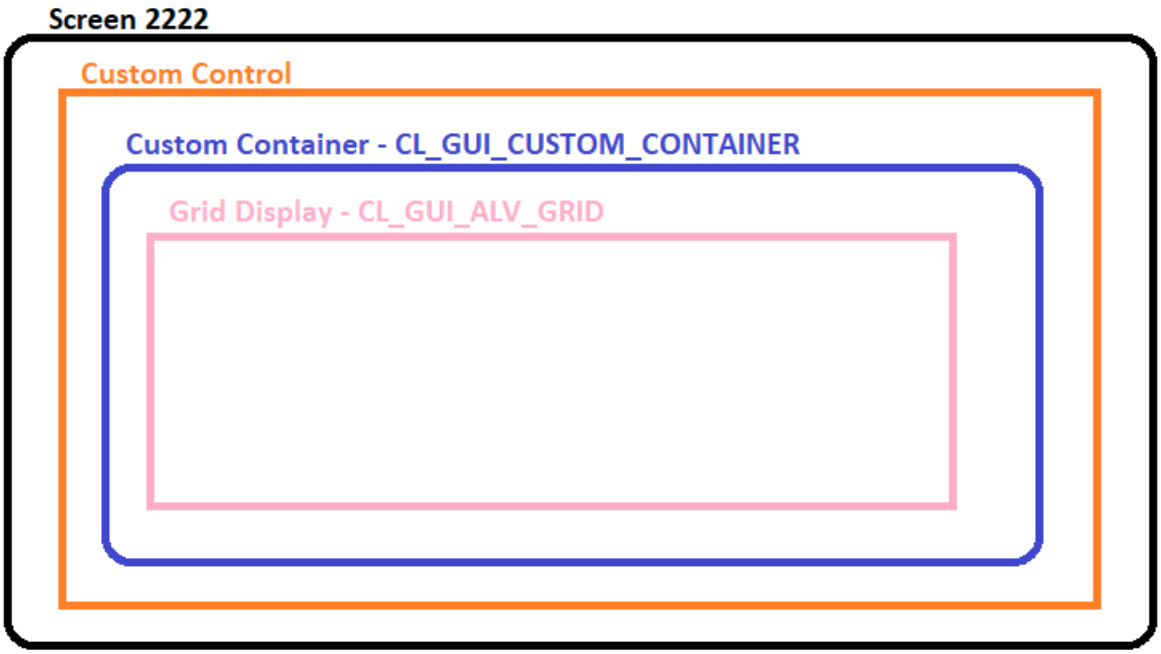
(here I'm maintaining normal method and event handler method in the same class).

```
REPORT ztest_2931.

DATA obj_t1 TYPE REF TO zcl_t1.
PARAMETERS p_kunnr TYPE kunnr.

START-OF-SELECTION.
  CREATE OBJECT obj_t1.

set HANDLER obj_t1->ehm_vbak for obj_t1.
  CALL METHOD obj_t1->get_vbak
    EXPORTING
      i_kunnr = p_kunnr                 " Customer Number
    IMPORTING
      es_vbak = DATA(ls_vbak).                  " Sales Document: Header Data

  write:/ ls_vbak-vbeln, ls_vbak-kunnr.
```

It will display the message with the customer number.

OOPS ALV

If we want to work on OOPS ALV then we need to create one screen, create one custom control in it. Custom control we can not use it directly, so refer it to custom container. From Custom Container, we can not display the data, so refer it to grid display.



CL_GUI_CUSTOM_CONTAINER:- Global Class which refer the Custom Control screen element
CL_GUI_ALV_GRID:- Global class which refer the ALV Grid.
SET_TABLE_FOR_FIRST_DISPLAY :- This is a method in CL_GUI_ALV_GRID which is used to display the data.

If you want to call this method then we must create object for gird class. Whenever we create the object for gird class then automatically one constructor is triggered and asks the input as object name of container class. Whenever we create the object for container class then automatically one constructor is triggered and asks the input as container name.

Steps to work with OOPS ALV:-

1. Create the selection-screen / input fields.
2. Declare the data internal table and field catalog internal table
3. Create the reference to the container and grid class start-ofselection.
4. Call screen <screen_number> & place custom control component in it.

PBO of the screen: -

1. Design the back button
2. Create the object to the container and gird class
3. Fill the data internal table
4. Fill the field catalog
5. Call the SET_TABLE_FOR_FIRST_DISPLAY method

PAI of the screen

1. Logic of back button

➔Based on the given customer number, display the customer information from KNA1 table

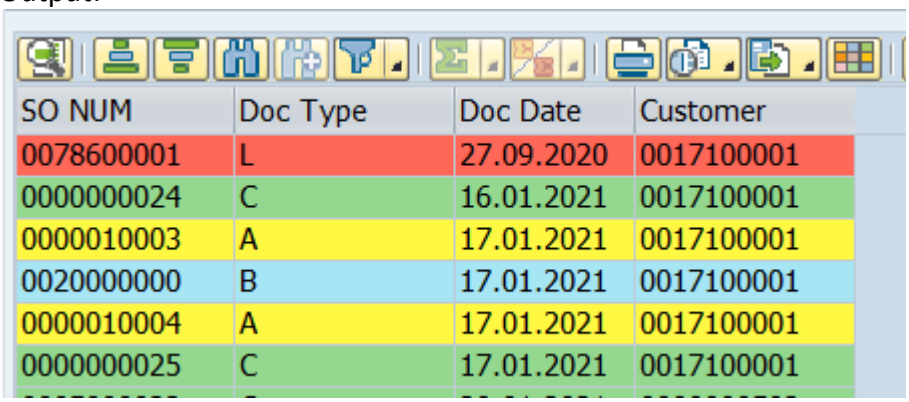
```
REPORT ztest_cust_info1.

DATA lv_kunnr TYPE kunnr.
DATA obj_cus_con TYPE REF TO cl_gui_custom_container.
DATA obj_grid TYPE REF TO cl_gui_alv_grid.
DATA lt_fcat TYPE lvc_t_fcat .
DATA ls_fcat LIKE LINE OF lt_fcat.
DATA lv_count TYPE i.
SELECT-OPTIONS s_kunnr FOR lv_kunnr.

START-OF-SELECTION.
CALL SCREEN 2222.
*&---------------------------------------------------------------------*
*& Module STATUS_2222 OUTPUT
*&---------------------------------------------------------------------*
*&
*&---------------------------------------------------------------------*
MODULE status_2222 OUTPUT.
 SET PF-STATUS 'PF'.

CREATE OBJECT obj_cus_con
  EXPORTING
    container_name  = 'CUST_CTRL'.

CREATE OBJECT obj_grid
  EXPORTING
    i_parent          = obj_cus_con.

SELECT kunnr, name1, ort01 FROM kna1 INTO TABLE @DATA(lt_kna1) WHERE kunnr IN
@s_kunnr.
  IF lt_kna1 IS NOT INITIAL.

  PERFORM fill_fcat USING 'KUNNR' 'Customer'.
  PERFORM fill_fcat USING 'NAME1' 'Customer Name'.
  PERFORM fill_fcat USING 'ORT01' 'City'.

    CALL METHOD obj_grid->set_table_for_first_display
      CHANGING
        it_outtab                     = lt_kna1
        it_fieldcatalog               = lt_fcat.
    ENDIF.
ENDMODULE.

FORM fill_fcat  USING    VALUE(p_fname)
                         VALUE(p_value).
  lv_count = lv_count + 1.
  ls_fcat-fieldname = p_fname.
  ls_fcat-col_pos = lv_count.
  ls_fcat-coltext = p_value.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.
ENDFORM.
*&---------------------------------------------------------------------*
*&      Module  USER_COMMAND_2222  INPUT
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
MODULE user_command_2222 INPUT.
IF sy-ucomm = 'BACK'.
  LEAVE PROGRAM.
  ENDIF.
ENDMODULE.
```

➔Based on the given customer number, display the Sales information from VBAK table and apply some colours to the output.

```
*&---------------------------------------------------------------------*
*& Report ZTEST_SO
*&---------------------------------------------------------------------*
*&
*&---------------------------------------------------------------------*
REPORT ztest_so.
TYPES: BEGIN OF ty_so,
         vbeln TYPE vbak-vbeln,
         vbtyp TYPE vbak-vbtyp,
         audat TYPE vbak-audat,
         kunnr TYPE kunnr,
         col   TYPE char4,
       END OF ty_so.

DATA: lv_kunnr  TYPE kunnr,
      obj_so_cc TYPE REF TO cl_gui_custom_container,
      obj_so_gr TYPE REF TO cl_gui_alv_grid,
      lt_so     TYPE TABLE OF ty_so,
      ls_so     TYPE ty_so,
      lt_fcat   TYPE lvc_t_fcat,
      ls_fcat   TYPE lvc_s_fcat,
      ls_layout TYPE lvc_s_layo.

SELECT-OPTIONS s_kunnr FOR lv_kunnr.

START-OF-SELECTION.
  CALL SCREEN 1111.
*&---------------------------------------------------------------------*
*& Module STATUS_1111 OUTPUT
*&---------------------------------------------------------------------*
*&
*&---------------------------------------------------------------------*
MODULE status_1111 OUTPUT.
  SET PF-STATUS 'PF_SO'.
* SET TITLEBAR 'xxx'.

  CREATE OBJECT obj_so_cc
    EXPORTING
      container_name = 'CC_SO'.

  CREATE OBJECT obj_so_gr
    EXPORTING
      i_parent = obj_so_cc.

  SELECT vbeln
         vbtyp
         audat
         kunnr
    FROM vbak INTO TABLE lt_so
    WHERE kunnr IN s_kunnr.

  LOOP AT lt_so ASSIGNING FIELD-SYMBOL(<fs_so>).
    CASE <fs_so>-vbtyp.
      WHEN 'A'.
        <fs_so>-col = 'C310'.
      WHEN 'B'.
        <fs_so>-col = 'C410'.
      WHEN 'C'.
        <fs_so>-col = 'C510'.
      WHEN OTHERS.
        <fs_so>-col = 'C610'.
```

```
    ENDCASE.
  ENDLOOP.

  PERFORM fill_fieldcatalog.

  ls_layout-zebra = 'X'.
  ls_layout-info_fname = 'COL'.



  CALL METHOD obj_so_gr->set_table_for_first_display
    EXPORTING
      is_layout       = ls_layout                  " Layout
    CHANGING
      it_outtab       = lt_so                  " Output Table
      it_fieldcatalog = lt_fcat .                  " Field Catalog

ENDMODULE.

FORM fill_fieldcatalog .
  ls_fcat-fieldname = 'VBELN'.
  ls_fcat-col_pos = 1.
  ls_fcat-coltext = 'SO NUM'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.
  ls_fcat-fieldname = 'VBTYP'.
  ls_fcat-col_pos = 2.
  ls_fcat-coltext = 'Doc Type'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.
  ls_fcat-fieldname = 'AUDAT'.
  ls_fcat-col_pos = 3.
  ls_fcat-coltext = 'Doc Date'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.
  ls_fcat-fieldname = 'KUNNR'.
  ls_fcat-col_pos = 4.
  ls_fcat-coltext = 'Customer'.
*  LS_FCAT-emphasize = 'C310'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.
ENDFORM.

MODULE user_command_1111 INPUT.
  IF sy-ucomm = 'BACK'.
    LEAVE PROGRAM.
  ENDIF.
ENDMODULE.
```

Output:



SO NUM	Doc Type	Doc Date	Customer
0078600001	L	27.09.2020	0017100001
0000000024	C	16.01.2021	0017100001
0000010003	A	17.01.2021	0017100001
0020000000	B	17.01.2021	0017100001
0000010004	A	17.01.2021	0017100001
0000000025	C	17.01.2021	0017100001

➔Based on the given customer number, display the Sales information from VBAK table and apply some colours for particular one column in the output.

```
*&---------------------------------------------------------------------*
*& Report ZTEST_SO
*&---------------------------------------------------------------------*
*&
*&---------------------------------------------------------------------*
REPORT ztest_so1.
TYPES: BEGIN OF ty_so,
         vbeln TYPE vbak-vbeln,
         vbtyp TYPE vbak-vbtyp,
         audat TYPE vbak-audat,
         kunnr TYPE kunnr,
         col   TYPE lvc_t_scol,
       END OF ty_so.

DATA: lv_kunnr  TYPE kunnr,
      obj_so_cc TYPE REF TO cl_gui_custom_container,
      obj_so_gr TYPE REF TO cl_gui_alv_grid,
      lt_so     TYPE TABLE OF ty_so,
      ls_so     TYPE ty_so,
      lt_fcat   TYPE lvc_t_fcat,
      ls_fcat   TYPE lvc_s_fcat,
      ls_layout TYPE lvc_s_layo,
      ls_scol TYPE lvc_s_scol.

SELECT-OPTIONS s_kunnr FOR lv_kunnr.

START-OF-SELECTION.
  CALL SCREEN 1111.

MODULE status_1111 OUTPUT.
  SET PF-STATUS 'PF_SO'.

  CREATE OBJECT obj_so_cc
    EXPORTING
      container_name = 'CC_SO'.

  CREATE OBJECT obj_so_gr
    EXPORTING
      i_parent = obj_so_cc.

  SELECT vbeln
         vbtyp
         audat
         kunnr
    FROM vbak INTO CORRESPONDING FIELDS OF TABLE lt_so
    WHERE kunnr IN s_kunnr.

  LOOP AT lt_so ASSIGNING FIELD-SYMBOL(<fs_so>).
    CASE <fs_so>-vbtyp.
      WHEN 'A'.
        ls_scol-fname = 'VBELN'.
        ls_scol-color-col = '3'.
        ls_scol-color-int = '1'.
        ls_scol-color-inv = '0'.
        APPEND ls_scol TO <fs_so>-col.
        CLEAR ls_scol.
      WHEN 'B'.
        ls_scol-fname = 'VBELN'.
        ls_scol-color-col = '4'.
        ls_scol-color-int = '1'.
        ls_scol-color-inv = '0'.
        APPEND ls_scol TO <fs_so>-col.
        CLEAR ls_scol.
```

```
      WHEN 'C'.
        ls_scol-fname = 'VBELN'.
        ls_scol-color-col = '5'.
        ls_scol-color-int = '1'.
        ls_scol-color-inv = '0'.
        APPEND ls_scol TO <fs_so>-col.
        CLEAR ls_scol.
      WHEN OTHERS.
        ls_scol-fname = 'VBELN'.
        ls_scol-color-col = '6'.
        ls_scol-color-int = '1'.
        ls_scol-color-inv = '0'.
        APPEND ls_scol TO <fs_so>-col.
        CLEAR ls_scol.
    ENDCASE.
  ENDLOOP.

  PERFORM fill_fieldcatalog.

  ls_layout-zebra = 'X'.
  ls_layout-ctab_fname = 'COL'.


  CALL METHOD obj_so_gr->set_table_for_first_display
    EXPORTING
      is_layout       = ls_layout                  " Layout
    CHANGING
      it_outtab       = lt_so                  " Output Table
      it_fieldcatalog = lt_fcat .                  " Field Catalog

ENDMODULE.

FORM fill_fieldcatalog .
  ls_fcat-fieldname = 'VBELN'.
  ls_fcat-col_pos = 1.
  ls_fcat-coltext = 'SO NUM'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.

  ls_fcat-fieldname = 'VBTYP'.
  ls_fcat-col_pos = 2.
  ls_fcat-coltext = 'Doc Type'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.

  ls_fcat-fieldname = 'AUDAT'.
  ls_fcat-col_pos = 3.
  ls_fcat-coltext = 'Doc Date'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.

  ls_fcat-fieldname = 'KUNNR'.
  ls_fcat-col_pos = 4.
  ls_fcat-coltext = 'Customer'.
*  LS_FCAT-emphasize = 'C310'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.

ENDFORM.

MODULE user_command_1111 INPUT.
  IF sy-ucomm = 'BACK'.
    LEAVE PROGRAM.
  ENDIF.
ENDMODULE.
```

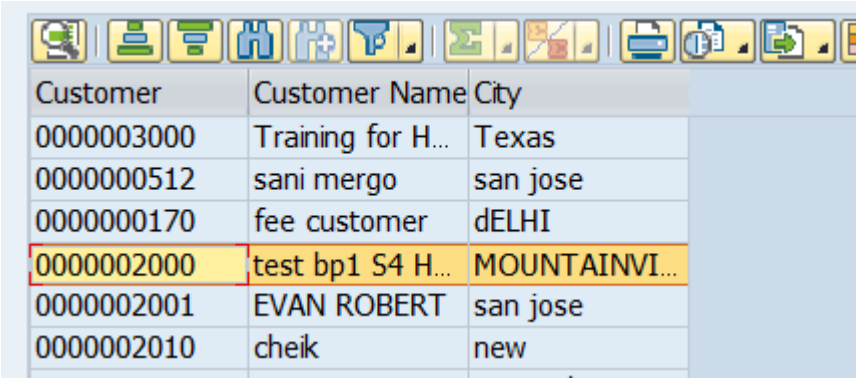
SO NUM	Doc Type	Doc Date	Customer
0000000023	C	16.01.2021	0017100001
0078600001	L	27.09.2020	0017100001
0000000024	C	16.01.2021	0017100001
0000010003	A	17.01.2021	0017100001
0020000000	B	17.01.2021	0017100001
0000010004	A	17.01.2021	0017100001
0000000025	C	17.01.2021	0017100001

➔Based on the given customer number, display the customer information from KNA1 table. If double click on customer number, display the sales orders related to that customer in the next screen.

```
REPORT ztest_cust_info2.
TYPES: BEGIN OF ty_kna1,
         kunnr TYPE kunnr,
         name1 TYPE name1,
         ort01 TYPE ort01,
       END OF ty_kna1.

DATA lv_kunnr TYPE kunnr.
DATA obj_cus_con TYPE REF TO cl_gui_custom_container.
DATA obj_grid TYPE REF TO cl_gui_alv_grid.
DATA lt_fcat TYPE lvc_t_fcat .
DATA ls_fcat LIKE LINE OF lt_fcat.
DATA lv_count TYPE i.

DATA lt_fcat_so TYPE lvc_t_fcat.
DATA ls_fcat_so TYPE lvc_s_fcat.

DATA obj_cc_so TYPE REF TO cl_gui_custom_container.
DATA obj_grid_so TYPE REF TO cl_gui_alv_grid.

DATA lt_kna1 TYPE TABLE OF ty_kna1.
DATA ls_kna1 TYPE ty_kna1.

SELECT-OPTIONS s_kunnr FOR lv_kunnr.

CLASS lcl_so DEFINITION.
  PUBLIC SECTION.
    METHODS: ehm_so FOR EVENT double_click OF cl_gui_alv_grid IMPORTING e_row
e_column.
ENDCLASS.

CLASS lcl_so IMPLEMENTATION.
  METHOD ehm_so.
    IF e_column-fieldname = 'KUNNR'.
      READ TABLE lt_kna1 INTO ls_kna1 INDEX e_row-index.
      CALL SCREEN 2223.
    ENDIF.
  ENDMETHOD.
ENDCLASS.

START-OF-SELECTION.
```

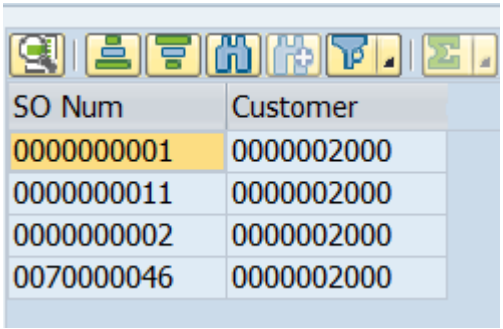
```
  DATA obj_so TYPE REF TO lcl_so.
  CREATE OBJECT obj_so.

  CALL SCREEN 2222.

MODULE status_2222 OUTPUT.
  SET PF-STATUS 'PF'.

  CREATE OBJECT obj_cus_con
    EXPORTING
      container_name = 'CUST_CTRL'.

  CREATE OBJECT obj_grid
    EXPORTING
      i_parent = obj_cus_con.

  SELECT kunnr name1 ort01 FROM kna1 INTO TABLE lt_kna1 WHERE kunnr IN s_kunnr.

  IF lt_kna1 IS NOT INITIAL.
    PERFORM fill_fcat USING 'KUNNR' 'Customer'.
    PERFORM fill_fcat USING 'NAME1' 'Customer Name'.
    PERFORM fill_fcat USING 'ORT01' 'City'.

    SET HANDLER obj_so->ehm_so FOR obj_grid.

    CALL METHOD obj_grid->set_table_for_first_display
      CHANGING
        it_outtab       = lt_kna1
        it_fieldcatalog = lt_fcat.
  ENDIF.

ENDMODULE.

FORM fill_fcat  USING    VALUE(p_fname)
                         VALUE(p_value).
  lv_count = lv_count + 1.
  ls_fcat-fieldname = p_fname.
  ls_fcat-col_pos = lv_count.
  ls_fcat-coltext = p_value.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.
ENDFORM.

MODULE user_command_2222 INPUT.
  IF sy-ucomm = 'BACK'.
    LEAVE PROGRAM.
  ENDIF.
ENDMODULE.

MODULE status_2223 OUTPUT.
  SET PF-STATUS 'PF_SO'.
* SET TITLEBAR 'xxx'.

  CREATE OBJECT obj_cc_so
    EXPORTING
      container_name = 'CC_SO'.

  CREATE OBJECT obj_grid_so
    EXPORTING
      i_parent = obj_cc_so.

  SELECT vbeln, kunnr FROM vbak INTO TABLE @DATA(lt_vbak) WHERE kunnr =
@ls_kna1-kunnr.

  ls_fcat_so-fieldname = 'VBELN'.
```

```
  ls_fcat_so-col_pos = 1.
  ls_fcat_so-coltext = 'SO Num'.
  APPEND ls_fcat_so TO lt_fcat_so.
  CLEAR ls_fcat_so.

  ls_fcat_so-fieldname = 'KUNNR'.
  ls_fcat_so-col_pos = 2.
  ls_fcat_so-coltext = 'Customer'.
  APPEND ls_fcat_so TO lt_fcat_so.
  CLEAR ls_fcat_so.


  CALL METHOD obj_grid_so->set_table_for_first_display
    CHANGING
      it_outtab       = lt_vbak                  " Output Table
      it_fieldcatalog = lt_fcat_so.                " Field Catalog


ENDMODULE.

MODULE user_command_2223 INPUT.
  IF sy-ucomm = 'BACK'.
    LEAVE TO SCREEN 0.
  ENDIF.
ENDMODULE.
```



Customer	Customer Name	City
0000003000	Training for H...	Texas
0000000512	sani mergo	san jose
0000000170	fee customer	dELHI
0000002000	test bp1 S4 H...	MOUNTAINVI...
0000002001	EVAN ROBERT	san jose
0000002010	cheik	new

If we double click on customer 2000, sales information should display in the second screen.



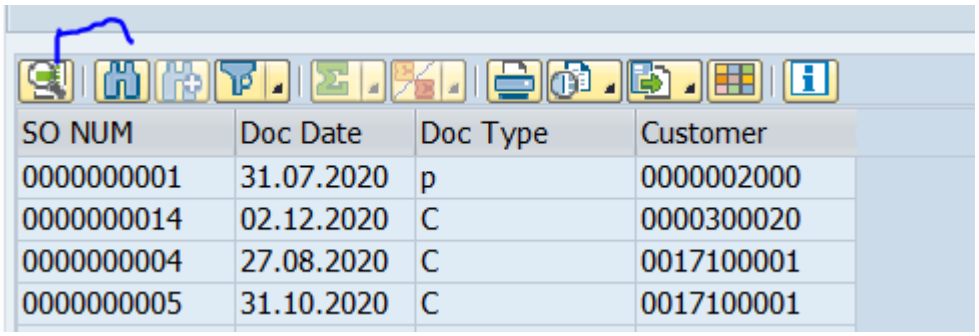
SO Num	Customer
0000000001	0000002000
0000000011	0000002000
0000000002	0000002000
0070000046	0000002000

➔Based on the given sales document number, fetch the sales information. Remove the ascending and descending options from standard tool bar.

```
REPORT ztest_so2.
TYPES: BEGIN OF ty_so,
         vbeln TYPE vbeln_va,
         vbtyp TYPE vbtyp,
         audat TYPE audat,
         kunnr TYPE kunnr,
       END OF ty_so.

DATA: lv_vbeln  TYPE vbak-vbeln,
```

```
      obj_so_cc TYPE REF TO cl_gui_custom_container,
      obj_so_gr TYPE REF TO cl_gui_alv_grid,
      lt_fcat   TYPE lvc_t_fcat,
      ls_fcat   TYPE lvc_s_fcat,
      lt_so     TYPE TABLE OF ty_so,
      ls_so     TYPE ty_so,
      lt_tool   TYPE ui_functions,
      ls_tool   TYPE ui_func.

SELECT-OPTIONS s_vbeln FOR lv_vbeln.

START-OF-SELECTION.
  CALL SCREEN 1234.

MODULE status_1234 OUTPUT.
  SET PF-STATUS 'PF_SO'.
  CREATE OBJECT obj_so_cc
    EXPORTING
      container_name = 'CC_SO'.

  CREATE OBJECT obj_so_gr
    EXPORTING
      i_parent = obj_so_cc.

  SELECT vbeln
         vbtyp
         audat
         kunnr
    FROM vbak INTO TABLE lt_so
    WHERE vbeln IN s_vbeln.

  PERFORM fill_fcat.  "Fill field catalog
  PERFORM toobar_excl. "Toolbar excluding icons list

  CALL METHOD obj_so_gr->set_table_for_first_display
    EXPORTING
      it_toolbar_excluding = lt_tool                  " Excluded Toolbar Standard
Functions
    CHANGING
      it_outtab            = lt_so                 " Output Table
      it_fieldcatalog      = lt_fcat.              " Field Catalog

ENDMODULE.

FORM fill_fcat.
  ls_fcat-fieldname = 'VBELN'.
  ls_fcat-col_pos = 1.
  ls_fcat-coltext = 'SO NUM'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.

  ls_fcat-fieldname = 'AUDAT'.
  ls_fcat-col_pos = 2.
  ls_fcat-coltext = 'Doc Date'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.

  ls_fcat-fieldname = 'VBTYP'.
  ls_fcat-col_pos = 3.
  ls_fcat-coltext = 'Doc Type'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.

  ls_fcat-fieldname = 'KUNNR'.
  ls_fcat-col_pos = 4.
```

```
  ls_fcat-coltext = 'Customer'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.
ENDFORM.

FORM toobar_excl.
  ls_tool = cl_gui_alv_grid=>mc_fc_sort_asc.
  APPEND ls_tool TO lt_tool.
  ls_tool = cl_gui_alv_grid=>mc_fc_sort_dsc.
  APPEND ls_tool TO lt_tool.

ENDFORM.

MODULE user_command_1234 INPUT.
  IF sy-ucomm = 'BACK'.
    LEAVE PROGRAM.
  ENDIF.
ENDMODULE.
```



SO NUM	Doc Date	Doc Type	Customer
0000000001	31.07.2020	p	0000002000
0000000014	02.12.2020	C	0000300020
0000000004	27.08.2020	C	0017100001
0000000005	31.10.2020	C	0017100001

➔Based on the given sales document number, fetch Sales and Billing information and display both information in the single screen.

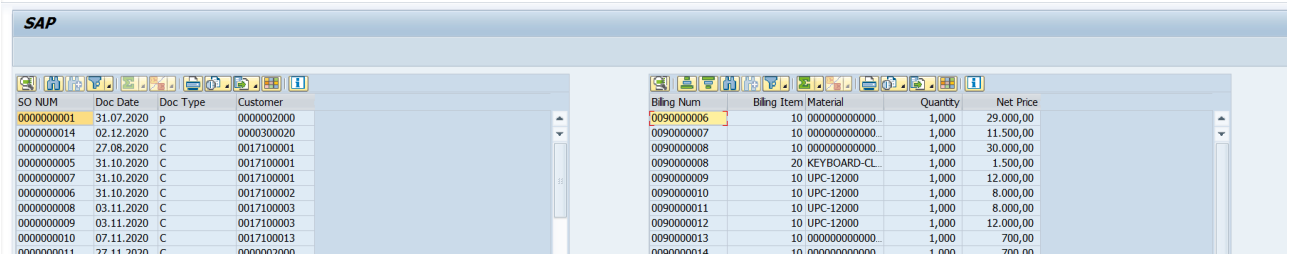
```
REPORT ztest_so2.
TYPES: BEGIN OF ty_so,
         vbeln TYPE vbeln_va,
         vbtyp TYPE vbtyp,
         audat TYPE audat,
         kunnr TYPE kunnr,
       END OF ty_so,

       BEGIN OF ty_bill,
         vbeln TYPE vbrk-vbeln,
         posnr TYPE vbrp-posnr,
         matnr TYPE matnr,
         fkimg TYPE fkimg,
         netwr TYPE vbrp-netwr,
       END OF ty_bill.

DATA: lv_vbeln  TYPE vbak-vbeln,
      obj_so_cc TYPE REF TO cl_gui_custom_container,
      obj_so_gr TYPE REF TO cl_gui_alv_grid,
      obj_bl_cc TYPE REF TO cl_gui_custom_container,
      obj_bl_gr TYPE REF TO cl_gui_alv_grid,
      lt_fcat   TYPE lvc_t_fcat,
      ls_fcat   TYPE lvc_s_fcat,
      lt_fcat2  TYPE lvc_t_fcat,
      ls_fcat2  TYPE lvc_s_fcat,
      lt_so     TYPE TABLE OF ty_so,
      ls_so     TYPE ty_so,
      lt_bill   TYPE TABLE OF ty_bill,
      ls_bill   TYPE ty_bill,
```

```
      lt_tool   TYPE ui_functions,
      ls_tool   TYPE ui_func.

SELECT-OPTIONS s_vbeln FOR lv_vbeln.

START-OF-SELECTION.
  CALL SCREEN 1234.
MODULE status_1234 OUTPUT.
  SET PF-STATUS 'PF_SO'.
  CREATE OBJECT obj_so_cc
    EXPORTING
      container_name = 'CC_SO'.

  CREATE OBJECT obj_so_gr
    EXPORTING
      i_parent = obj_so_cc.

  CREATE OBJECT obj_bl_cc
    EXPORTING
      container_name = 'CC_BILL'.

  CREATE OBJECT obj_bl_gr
    EXPORTING
      i_parent = obj_bl_cc.

  SELECT vbeln
         vbtyp
         audat
         kunnr
    FROM vbak INTO TABLE lt_so
    WHERE vbeln IN s_vbeln.

  IF lt_so IS NOT INITIAL.
    SELECT vbeln,
           posnr
      FROM vbap INTO TABLE @DATA(lt_vbap)
      WHERE vbeln IN @s_vbeln.

    IF lt_vbap IS NOT INITIAL.
      SELECT vbeln
             posnr
             matnr
             fkimg
             netwr
        FROM vbrp INTO TABLE lt_bill
        FOR ALL ENTRIES IN lt_vbap
        WHERE aubel = lt_vbap-vbeln
        AND aupos = lt_vbap-posnr.
    ENDIF.
  ENDIF.


  PERFORM fill_fcat.  "Fill field catalog
  PERFORM toobar_excl. "Toolbar excluding icons list

  CALL METHOD obj_so_gr->set_table_for_first_display
    EXPORTING
      it_toolbar_excluding = lt_tool                  " Excluded Toolbar Standard
Functions
    CHANGING
      it_outtab            = lt_so                 " Output Table
      it_fieldcatalog      = lt_fcat.              " Field Catalog

  CALL METHOD obj_bl_gr->set_table_for_first_display
    CHANGING
```

```
      it_outtab       = lt_bill                    " Output Table
      it_fieldcatalog = lt_fcat2.                   " Field Catalog
ENDMODULE.

FORM fill_fcat.
*SO field catalog
  ls_fcat-fieldname = 'VBELN'.
  ls_fcat-col_pos = 1.
  ls_fcat-coltext = 'SO NUM'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.

  ls_fcat-fieldname = 'AUDAT'.
  ls_fcat-col_pos = 2.
  ls_fcat-coltext = 'Doc Date'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.

  ls_fcat-fieldname = 'VBTYP'.
  ls_fcat-col_pos = 3.
  ls_fcat-coltext = 'Doc Type'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.

  ls_fcat-fieldname = 'KUNNR'.
  ls_fcat-col_pos = 4.
  ls_fcat-coltext = 'Customer'.
  APPEND ls_fcat TO lt_fcat.
  CLEAR ls_fcat.

*Billing Fieldcatalog
  ls_fcat2-fieldname = 'VBELN'.
  ls_fcat2-col_pos = 1.
  ls_fcat2-coltext = 'Billing Num'.
  APPEND ls_fcat2 TO lt_fcat2.
  CLEAR ls_fcat2.

  ls_fcat2-fieldname = 'POSNR'.
  ls_fcat2-col_pos = 2.
  ls_fcat2-coltext = 'Billing Item'.
  APPEND ls_fcat2 TO lt_fcat2.
  CLEAR ls_fcat2.

  ls_fcat2-fieldname = 'MATNR'.
  ls_fcat2-col_pos = 3.
  ls_fcat2-coltext = 'Material'.
  APPEND ls_fcat2 TO lt_fcat2.
  CLEAR ls_fcat2.

  ls_fcat2-fieldname = 'FKIMG'.
  ls_fcat2-col_pos = 4.
  ls_fcat2-coltext = 'Quantity'.
  APPEND ls_fcat2 TO lt_fcat2.
  CLEAR ls_fcat2.

  ls_fcat2-fieldname = 'NETWR'.
  ls_fcat2-col_pos = 5.
  ls_fcat2-coltext = 'Net Price'.
  APPEND ls_fcat2 TO lt_fcat2.
  CLEAR ls_fcat2.
ENDFORM.

FORM toobar_excl.
  ls_tool = cl_gui_alv_grid=>mc_fc_sort_asc.
  APPEND ls_tool TO lt_tool.
```

```
  ls_tool = cl_gui_alv_grid=>mc_fc_sort_dsc.
  APPEND ls_tool TO lt_tool.
ENDFORM.

MODULE user_command_1234 INPUT.
  IF sy-ucomm = 'BACK'.
    LEAVE PROGRAM.
  ENDIF.
ENDMODULE.
```

Output:



SO NUM	Doc Date	Doc Type	Customer
0000000001	31.07.2020	p	0000002000
0000000014	02.12.2020	C	0000300020
0000000004	27.08.2020	C	0017100001
0000000005	31.10.2020	C	0017100001
0000000007	31.10.2020	C	0017100001
0000000006	31.10.2020	C	0017100002
0000000008	03.11.2020	C	0017100003
0000000009	03.11.2020	C	0017100003
0000000010	07.11.2020	C	0017100013

Biling Num	Biling Item	Material	Quantity	Net Price
0090000006	10	000000000000...	1,000	29.000,00
0090000007	10	000000000000...	1,000	11.500,00
0090000008	10	000000000000...	1,000	30.000,00
0090000008	20	KEYBOARD-CL...	1,000	1.500,00
0090000009	10	UPC-12000	1,000	12.000,00
0090000010	10	UPC-12000	1,000	8.000,00
0090000011	10	UPC-12000	1,000	8.000,00
0090000012	10	UPC-12000	1,000	12.000,00
0090000013	10	000000000000...	1,000	700,00

Hi friends….

Please read the bellow message whenever you are free. Today we have food, clothes, shelter. Because of we are employees. But lot of people don't have these much of facilities. I mean orphan children, handicapped children, destitute senior citizens are suffering with minimum facilities. A few children are begging in the bus stops, platforms… a few people will left their old age parents at bus stops also. Orphan homes or Old age home people take care of them. A few NGOs are running their organizations without providing the food three times per day. They can provide when they have fund. In remaining days they may provide 1 time or 2 times per day.

A few people will think like, "I want to help to the orphan children or handicapped children. But is it possible only with me… How can I provide that much of money for them??" They may have these type of doubts. So we started one NGO named as SHREE JANANI FOUNDATION. Collected fund will spend for orphan children / handicapped children / old age people. These NGO will provide something whatever they don't have. But this NGO never give 1 rupee also for them, just provide their necessity.

If you want to join with us, you can call 8332999399 or send mail to helpdesk@shreejanani.org. You can help orphan children from SHREE JANANI FOUNDATION.

General Secretary

M N Sathish Kumar Reddy

Mobile: 9866079202

Mail: satishkumarreddy.mn@gmail.com