

The **Presentation layer** consists of any input device that can be used to control SAP system. This could be a web browser, a mobile device and so on. All the central processing takes place in **Application server**.

The Application server is not just one system in itself, but it can be multiple instances of the processing system.

The server communicates with the **Database layer** that is usually kept on a separate server, mainly for performance reasons and also for security. Communication happens between each layer of the system, from the Presentation layer to the Database and then back up the chain.

DATA TYPES

1. Data types can be defined or pre-defined types.
2. Pre-defined data types are also called elementary in sap karnel.
3. User defined data types are defined ‘TYPE’ keyword.
4. All the elementary types are C, N, D, P, X, I,F, DECFLOAT 16 OR 34.

In SAP ABAP (Advanced Business Application Programming), data types define the kind of data that can be stored in variables or fields within a program. ABAP provides several predefined data types that cover a wide range of data requirements. Here are some commonly used data types in SAP ABAP:

**Elementary Data Types:**

* Character (CHAR): Used to store single characters or strings of characters.
* Numeric (NUMC): Used for numeric values, such as integers or floating-point numbers.
* Date (DATS): Used to store dates.
* Time (TIMS): Used to store times.
* Binary (RAW): Used for storing binary data, such as images or files.

**Reference Data Types:**

* Integer (INT): Used for whole numbers.
* Float (FLOAT): Used for floating-point numbers.
* String (STRING): Used to store character strings.
* Boolean (BOOL): Used to represent true or false values.
* Decimal (DEC): Used for decimal numbers with high precision.

**Internal Table Types:**

* Standard Table (TABLE): Used for storing internal tables that have a fixed structure with a single key.
* Sorted Table (SORTED TABLE): Similar to a standard table but automatically keeps the data sorted according to a key field.
* Hashed Table (HASHED TABLE): Stores data in a hashed format, allowing for fast access based on a key field.

Structure Data Types:

* Structure (STRUCTURE): Used to define a custom data type consisting of multiple fields.
* Data Element (DATA): A predefined structure that can be used to define fields in a structure.

**DATA OBJECTS**

Data objects are physical memory unit that is defined in ABAP program to store the data.

Data objects are local to the program.

Data objects are derived from technical attributes from data type.

**ABAP SYNTAX**

1. An ABAP program consist of individual ABAP statements. Each statement begins with a keyword and ends with a period.
2. The first ABAP word is known as a keyword. The other words are either operands, operators or additions to keywords.
3. ABAP words must be separated by at least one space.
4. There can be more than one statement in a program line and a statement may extend over several program lines.
5. ABAP code is not case sensitive.

Example 1:-

**Parameters p\_name c Length 30.**

“Here parameter statement prompts the user to enter a value for this parameter when it is executed.

**Write ‘my name is:’**

“This line display the string it serves as text message.

**Write p\_name.**

“Here parameter on the screen it retrieves the value entered by the user when the program was executed.

**DATA** is a keyword that defines Data Object. If we want define data object we use Data Viewer.

Example 2:-

REPORT ZMAHE\_FIRST.  
Data v\_name Type c length 20. *"Here data type defines the Data Object. Type is used to specified the type of the data*  
  
v\_name = 'Mahesh'. *"Here we are storing the data in the object*  
WRITE: v\_name.

Or

REPORT ZMAHE\_FIRST.  
Data v\_name Type c length 20.  
Data f\_name TYPE c LENGTH 20. *"Here data type defines the Data Object. Type is used to specify the type of the data we defined length here*  
Data b\_name TYPE c LENGTH 20. *“Here we can give data for every field*   
  
v\_name = 'Mahesh'. *"Here we are storing the data in the object*  
WRITE: v\_name.

Or

REPORT ZMAHE\_FIRST.  
Data: v\_name Type c length 20,  
 f\_name TYPE c LENGTH 20, *"Here data type defines the Data Object. Type is used to specify the type of the data*  
 b\_name TYPE c LENGTH 20. *“Here we can give one data for every field using semicolon (:)*  
  
v\_name = 'Mahesh'. *"Here we are storing the data in the object*  
WRITE: v\_name.

Here every time we can’t change the length of the data. To clear this problem we can use the pre-defined data type.

The TYPE keyword defines Data type and DATA keyword defines the data of them.

TYPES ty\_user TYPE c LENGTH 20. *"Here this type refers the elementary data and ty\_user is not a memory it is data type. We can't store any data inn that.*  
  
ty\_user = 'Mahesh'. *"If we try to store the data it shows error it shows unknown.*

We can use as data object

TYPES ty\_user TYPE c LENGTH 20.  
  
Data: v\_name Type ty\_user,  
 f\_name TYPE ty\_user, *"Here ty\_user can use as data object*  
 b\_name TYPE ty\_user.   
  
  
v\_name = 'Mahesh'. *"here we are storing the data in the object*  
WRITE: v\_name.

What wherever we define data in particular program they are local we can only access within a program.

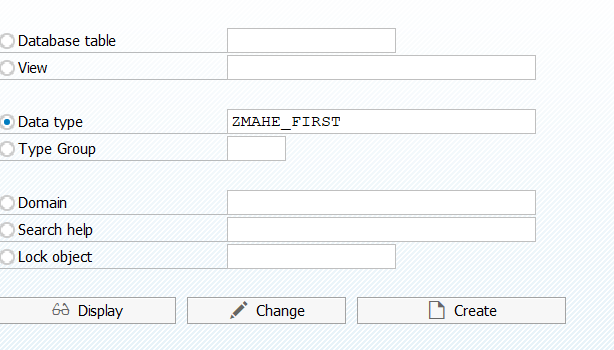
The global type data is called Data Element.

………………………………………………………………………………………………

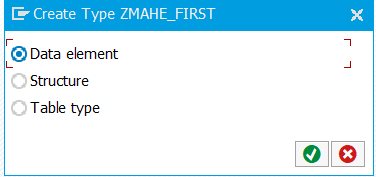
Elementary Data Types

To Create pre-defined data type we should follow some steps. Here they are

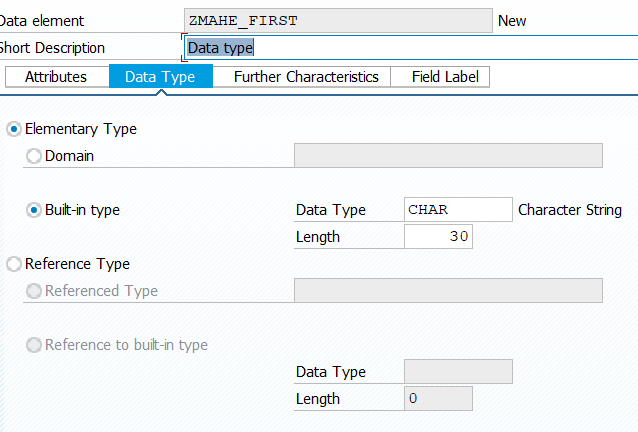
Step 1:-



Step 2:-



Step 3:-



Step 4:-

Activate

Here we are using pre-defined data in this program.

REPORT ZMAHE\_FIRST.  
  
TYPES zmahe\_first TYPE c LENGTH 20.  
  
Data: v\_name Type zmahe\_first,  
 f\_name TYPE zmahe\_first, *"Here zmahe\_first can use as data object*  
 b\_name TYPE zmahe\_first.  
  
  
v\_name = 'Mahesh'.  
f\_name = 'Babu'.  
b\_name = 'G'.  
  
WRITE: v\_name.  
WRITE: f\_name.  
WRITE: b\_name.

WE can use this data type centrally.

……………………………………………………………………………………………………………………………………………………………………………………………………

Domain:-

Here Domain is used to add multiple data elements so we can use single pre-defined data name to express different data type in program.

If we want to change the technical attributes we can edit in domain it will reflect in data elements.

Field

Structure

Table

**Structure**

Structure is a made-up of component type of any data type stores one another the other.

Types ty\_makt TYPE makt-matnr. *"Here we defining the data type*   
  
data v\_matnr TYPE ty\_makt. *"here we defining the data type to data object*

Example for structure.

REPORT ZMAHE\_FIRST.  
  
data v\_matnr TYPE makt-matnr. *“This work as single data element it stores single value*  
  
Data: BEGIN OF st\_makt, *"this work as structure but it stores single value*   
 matnr TYPE makt-matnr,  
 spras TYPE makt-spras,  
 maktx TYPE makt-maktx,  
 END OF st\_makt.  
  
v\_matnr = '100'.  
  
v\_matnr = '200'. *"When we declare 100 and 200 it replaces the first value it can't store more than one value it replace the data*  
write v\_matnr.  
  
St\_makt-matnr = '101'.  
  
  
st\_makt-matnr = '201'. *"When we declare 100 and 200 it replaces the first value it can't store more than one value it replace the data*  
WRITE st\_makt-matnr.

Example 1:-

Write a program to show the Material Number, Language and Material Description using Material Number.

REPORT ZMAHE\_FIRST.  
  
PARAMETERS v\_matnr TYPE makt-matnr.  
  
TYPES: BEGIN OF ty\_makt, *"Here we are refer the structure type using Types keyword*  
 matnr TYPE makt-matnr,  
 spras TYPE makt-spras,  
 maktx TYPE makt-maktx,  
 END OF ty\_makt.  
  
 data st\_makt TYPE ty\_makt. *"Here it defines the structure.*  
  
Select matnr spras maktx from makt into st\_makt where matnr eq v\_matnr. *"Here we retrieve the data of each field using select statement*   
  
WRITE: st\_makt-matnr.  
WRITE: st\_makt-spras. *"Here we display the data using Write statement*  
WRITE: st\_makt-maktx.  
ENDSELECT.

Or

Select \* from makt into st\_makt where matnr eq v\_matnr. “it shows error field is not long in st\_makt.

Or

Select matnr spras maktx from makt into st\_makt where matnr eq v\_matnr and spras eq 'E'. *"Here we retrevie the data of each field using select statement*

*Or*

Select \* from makt into st\_makt where matnr eq v\_matnr and spras eq 'E'. *"Here we retrevie the data of each field using select statement*

Example 2:- Write a program to show Material number ,plant name, storage location and material description using plant name.

PARAMETERS v\_werks TYPE mard-werks.  
  
TYPES: BEGIN OF ty\_mard,  
 matnr TYPE mard-matnr,  
 werks TYPE mard-werks,  
 lgort TYPE mard-lgort, *"here we take structure type for mard database.*  
 END OF ty\_mard.  
  
TYPES: BEGIN OF ty\_makt,  
 matnr TYPE makt-matnr, *"Here we take structure type for makt database*  
 maktx TYPE makt-maktx,  
 END OF ty\_makt.  
  
DATA st\_makt TYPE ty\_makt. *"Here we defining the structure for makt*  
  
DATA st\_mard TYPE ty\_mard. *"Here we defining the structure for mard*  
  
SELECT matnr werks lgort FROM mard INTO st\_mard WHERE werks EQ v\_werks. *"Here we retrieve the data from mard*   
 SELECT SINGLE matnr maktx FROM makt INTO st\_makt WHERE matnr EQ st\_mard-matnr AND spras EQ sy-langu. *"Here we retrieving the matnr and maktx data from makt here maktx is a material description and matnr is foreign key in this.*  
   
  
 WRITE:/ st\_mard-matnr,  
 st\_mard-werks,  
 st\_mard-lgort,  
 st\_makt-maktx.  
ENDSELECT.

……………………………………………………………………………………

Or

TYPES: BEGIN OF ty\_mard,  
 matnr TYPE mard-matnr,  
 werks TYPE mard-werks,  
 lgort TYPE mard-lgort,  
 END OF ty\_mard.  
  
  
DATA st\_mard TYPE ty\_mard.  
  
DATA v\_maktx TYPE makt-maktx. *"Here instead of structure type and field for makt directly we can specify the field name with database name*  
  
SELECT matnr werks lgort FROM mard INTO st\_mard WHERE werks EQ v\_werks.  
 SELECT SINGLE maktx FROM makt INTO v\_maktx WHERE matnr EQ st\_mard-matnr and spras EQ sy-langu.  
  
 WRITE: / st\_mard-matnr,  
 st\_mard-werks,  
 st\_mard-lgort,  
 v\_maktx.  
ENDSELECT.

**Example 3:- Write a program to show Material number ,plant number, storage location and material description plant name using plant number.**

Here to Write plant name we use T001w

PARAMETERS v\_werks TYPE mard-werks.  
  
TYPES: BEGIN OF ty\_mard,  
 matnr TYPE mard-matnr,  
 werks TYPE mard-werks,  
 lgort TYPE mard-lgort,  
 END OF ty\_mard.  
Data v\_name TYPE t001w-name1.  
  
DATA st\_mard TYPE ty\_mard.  
  
DATA v\_maktx TYPE makt-maktx. *"Here instead of structure type and field for makt directly we can specify the field name with database name*  
  
SELECT matnr werks lgort FROM mard INTO st\_mard WHERE werks EQ v\_werks.  
 SELECT SINGLE maktx FROM makt INTO v\_maktx WHERE matnr EQ st\_mard-matnr and spras eq sy-langu.  
 SELECT SINGLE name1 from t001w INTO v\_name where werks eq st\_mard-werks.  
  
  
 WRITE:/ st\_mard-matnr,  
 st\_mard-werks,  
 st\_mard-lgort,  
 v\_name,  
 v\_maktx.  
ENDSELECT.

………………………………………………………………………………………………………………………………………………………………………………………………………

TRANSCATION CODE

ME21N – CREATE EKKO - HEADER

ME22N – CHANGE EKPO - ITEM

ME23N – DISPLAY

Sales

VA01 – CREATE VBAK

VA02 – CHANGE VBAP

VA03 – DISPLAY

Delivery Document

VLO1 LIKP

VL02 LIPS

VL03

Billing

VF01 VBRK

VF02 VBRP

VF03

**To Create Table**

SE11 – database name – create – description – A-display – maintained Allowed

A

Data elements are

VBELN

FKDAT

KUNRG

NETWR

Technical settings

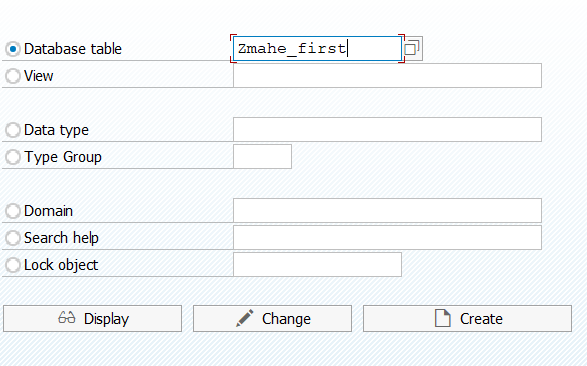
APPL1

Extra – Enhancement – can be enhanced – active

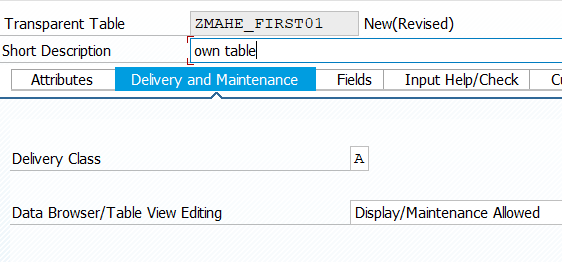
Utilities – base object – display or create.

Steps to Create the own table

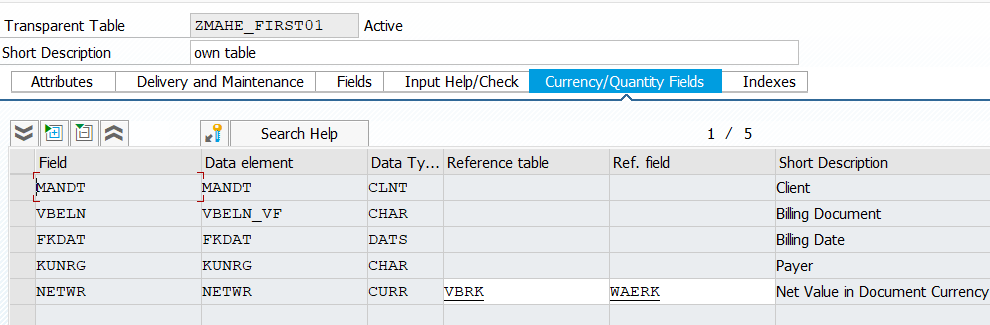
Step 1:-



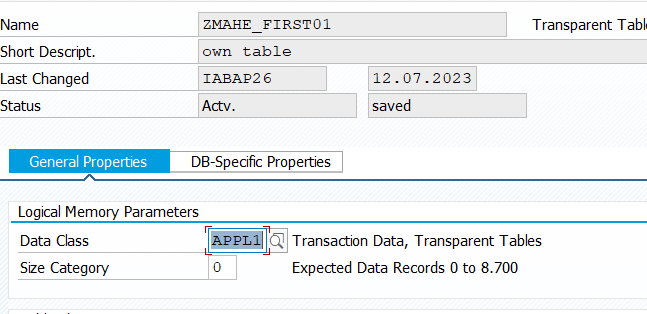
Step 2:



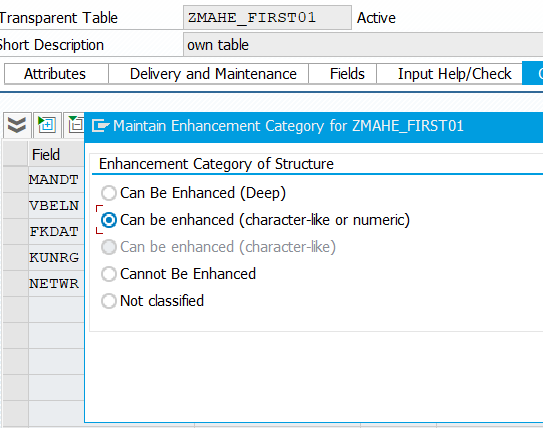
Step 3:



Step 4:-



Step 5:-



For item table we need

VBELN – VBELN-VF

POSNR – POSNR

FKIMG – FKIMG - VBRP MEINS

VRKME – VRKME

NETWR – NETWR - VBRK WAERK

Example 1:-

Write a program to

Doc no:

Payer:

Date:

Networth:

Item material quantity unit of measure netvalue

Here kna1 for payer and t006a

PARAMETERS v\_vbeln TYPE zmahe\_first01-vbeln.  
  
TYPES: BEGIN OF ty\_vbrk,  
 vbeln TYPE zmahe\_first01-vbeln,  
 fkdat TYPE zmahe\_first01-fkdat,  
 kunrg TYPE zmahe\_first01-kunrg,  
 netwr TYPE zmahe\_first01-netwr,  
 END OF ty\_vbrk.  
  
TYPES: BEGIN OF ty\_vbrp,  
 posnr TYPE zmahe\_first02-posnr,  
 fkimg TYPE zmahe\_first02-fkimg,  
 vrkme TYPE zmahe\_first02-vrkme,  
 netwr TYPE zmahe\_first02-netwr,  
 matnr TYPE zmahe\_first02-matnr,  
 END OF ty\_vbrp.  
  
DATA: st\_vbrk TYPE ty\_vbrk,  
 st\_vbrp TYPE ty\_vbrp,  
 v\_name TYPE kna1-name1,  
 v\_mesht TYPE t006a-mseht.  
  
  
select vbeln fkdat kunrg netwr FROM zmahe\_first01 INTO st\_vbrk WHERE vbeln EQ v\_vbeln.  
 SELECT SINGLE name1 FROM kna1 INTO v\_name WHERE kunnr EQ st\_vbrk-kunrg.  
 WRITE : /'DOCUMENT NO :', st\_vbrk-VBELN,  
 / 'Payer :', v\_name,  
 /'Date :', st\_vbrk-fkdat,  
 /'Net Value :', st\_vbrk-netwr left-justified.  
 ENDSELECT.  
 WRITE: / 'item',  
 40 'Material',  
 75 'Quantity',  
 90 'Units Of measure',  
 120 'Net Value'.  
  
 SELECT SINGLE posnr fkimg vrkme netwr matnr FROM zmahe\_first02 INTO st\_vbrp WHERE vbeln EQ v\_vbeln.  
 SELECT SINGLE mseht FROM t006a INTO v\_mesht WHERE msehi EQ st\_vbrp-vrkme.  
 WRITE: / st\_vbrp-posnr,  
 40 st\_vbrp-matnr,  
 75 st\_vbrp-fkimg left-justified,  
 90 v\_mesht,  
 120 st\_vbrp-netwr left-justified.

Example 2:-

PARAMETERS v\_vbeln TYPE zmahe\_first01-vbeln.  
  
*"TYPES: BEGIN OF ty\_vbrk,*  
*" vbeln TYPE zmahe\_first01-vbeln,*  
*" fkdat TYPE zmahe\_first01-fkdat,*  
*" kunrg TYPE zmahe\_first01-kunrg,*  
*" netwr TYPE zmahe\_first01-netwr,*  
*" END OF ty\_vbrk.*  
  
*"TYPES: BEGIN OF ty\_vbrp,*  
*" posnr TYPE zmahe\_first02-posnr,*  
*" fkimg TYPE zmahe\_first02-fkimg,*  
*" vrkme TYPE zmahe\_first02-vrkme,*  
*" netwr TYPE zmahe\_first02-netwr,*  
*" matnr TYPE zmahe\_first02-matnr,*  
*" END OF ty\_vbrp.*  
  
*"Here instead of Structure type we can refer the data with the help database name.*  
  
DATA st\_vbrk TYPE zmahe\_first01. *"Here we xan write structure like this by using database name*  
  
DATA st\_vbrp TYPE zmahe\_first02.  
  
  
DATA: v\_name TYPE kna1-name1,  
 v\_mesht TYPE t006a-mseht.  
  
  
SELECT \* FROM zmahe\_first01 INTO st\_vbrk WHERE vbeln EQ v\_vbeln.

“Here select statement takes all the fields instead of specifying separate fields we can write \*  
 SELECT SINGLE name1 FROM kna1 INTO v\_name WHERE kunnr EQ st\_vbrk-kunrg.  
 WRITE : /'DOCUMENT NO :', st\_vbrk-vbeln,  
 / 'Payer :', v\_name,  
 /'Date :', st\_vbrk-fkdat,  
 /'Net Value :', st\_vbrk-netwr LEFT-JUSTIFIED.  
ENDSELECT.  
WRITE: / 'item',  
 40 'Material',  
 75 'Quantity',  
 90 'Units Of measure',  
 120 'Net Value'.  
  
SELECT SINGLE \* FROM zmahe\_first02 INTO st\_vbrp WHERE vbeln EQ v\_vbeln.  
SELECT SINGLE mseht FROM t006a INTO v\_mesht WHERE msehi EQ st\_vbrp-vrkme.  
WRITE: / st\_vbrp-posnr,  
 40 st\_vbrp-matnr,  
 75 st\_vbrp-fkimg LEFT-JUSTIFIED,  
 90 v\_mesht,  
 120 st\_vbrp-netwr LEFT-JUSTIFIED.

Or

Instead of using the database name here we can use data type object to refer in the structure.

PARAMETERS v\_vbeln TYPE zmahe\_first01-vbeln.  
  
*"TYPES: BEGIN OF ty\_vbrk,*  
*" vbeln TYPE zmahe\_first01-vbeln,*  
*" fkdat TYPE zmahe\_first01-fkdat,*  
*" kunrg TYPE zmahe\_first01-kunrg,*  
*" netwr TYPE zmahe\_first01-netwr,*  
*" END OF ty\_vbrk.*  
  
*"TYPES: BEGIN OF ty\_vbrp,*  
*" posnr TYPE zmahe\_first02-posnr,*  
*" fkimg TYPE zmahe\_first02-fkimg,*  
*" vrkme TYPE zmahe\_first02-vrkme,*  
*" netwr TYPE zmahe\_first02-netwr,*  
*" matnr TYPE zmahe\_first02-matnr,*  
*" END OF ty\_vbrp.*  
  
*"Here instead of Structure type we can refer the data with the help database name.*  
  
  
*"DATA st\_vbrk TYPE zmahe\_first01. "Here we can write structure like this by using database name*  
  
*"DATA st\_vbrp TYPE zmahe\_first02.*  
  
DATA st\_vbrk TYPE zmahe\_vbrk.  
DATA st\_vbrp TYPE zmahe\_vbrp. *"Here instead of database name we can use data type object to refer the fields. We can refer local structure to Global structure.*  
  
DATA: v\_name TYPE kna1-name1,  
 v\_mesht TYPE t006a-mseht.  
  
  
SELECT \* FROM zmahe\_first01 INTO st\_vbrk WHERE vbeln EQ v\_vbeln.  
 SELECT SINGLE name1 FROM kna1 INTO v\_name WHERE kunnr EQ st\_vbrk-kunrg.  
 WRITE : /'DOCUMENT NO :', st\_vbrk-vbeln,  
 / 'Payer :', v\_name,  
 /'Date :', st\_vbrk-fkdat,  
 /'Net Value :', st\_vbrk-netwr LEFT-JUSTIFIED.  
ENDSELECT.  
WRITE: / 'item',  
 40 'Material',  
 75 'Quantity',  
 90 'Units Of measure',  
 120 'Net Value'.  
  
SELECT SINGLE \* FROM zmahe\_first02 INTO st\_vbrp WHERE vbeln EQ v\_vbeln.  
SELECT SINGLE mseht FROM t006a INTO v\_mesht WHERE msehi EQ st\_vbrp-vrkme.  
WRITE: / st\_vbrp-posnr,  
 40 st\_vbrp-matnr,  
 75 st\_vbrp-fkimg LEFT-JUSTIFIED,  
 90 v\_mesht,  
 120 st\_vbrp-netwr LEFT-JUSTIFIED.

……………………………………………………………………………………

Here Structure can only store one row of data. But Internal table which can store the multiple of data.

If you're using to process the internal table data, we call it a work area.

Ex: it\_vbrk type Table of ty\_vbrk,

Wa\_vbrk Like line of it\_vbrk.

What is Line Type?

The only tool is this structure and this Internet table should be of the same line type. The line type should be same.

TYPES: BEGIN OF ty\_makt,  
 matnr TYPE makt-matnr,  
 spras TYPE makt-spras,  
 maktx TYPE makt-maktx,  
 END OF ty\_makt.  
  
DATA st\_makt TYPE ty\_makt.  
  
Data: it\_makt TYPE TABLE of ty\_makt,  
 wa\_makt LIKE LINE OF it\_makt. *"It creates the internal table perspective can store one row internal table can store multiple rooms*  
  
*"here wa(work area) as structure for internal table*  
  
  
st\_makt matnr *" it is a structure when we refer st\_makt-matnr its refers only one row of data*  
it\_makt-matnr *"it is an internal type when we refer it\_makt-matnr its refers multiple row of data*

If data is in internal table how do we access.

To access the internal table. We use

1. Key
2. Index

All the rows in the inner table. If we want to access any one row or more than one.

Read access only one row.

Loop access more than one row.

Read and loop statement simply copies the data into the work period from the internal table to one area**.**

TYPES: BEGIN OF ty\_makt,  
 matnr TYPE makt-matnr,  
 spras TYPE makt-spras,  
 maktx TYPE makt-maktx,  
 END OF ty\_makt.  
  
DATA st\_makt TYPE ty\_makt.  
  
Data: it\_makt TYPE TABLE of ty\_makt,  
 wa\_makt LIKE LINE OF it\_makt. *"It creates the internal table perspective can store one row internal table can store multiple rooms*  
  
select matnr spras maktx from makt INTO TABLE it\_makt.  
  
 READ TABLE it\_makt INTO wa\_makt index 3. *"Using Read statement we are accessing the row 3 by referring index it only one row*  
 READ TABLE it\_makt INTO wa\_makt with key matnr = p\_matnr. *"Using Read statement here we accessing the data with key*  
  
 write: wa\_makt-matnr,  
 wa\_makt-spras,  
 wa\_makt-maktx.

To Access more than one row instead of using select and select statements we can use Loop statement.

Loop at it\_makt into wa\_makt.

Select matnr spras maktx from makt INTO TABLE it\_makt.  
 Loop at it\_makt INTO wa\_makt. *"Here instead of select statement we can use Loop statement*

Example 1:-

TYPES: BEGIN OF ty\_makt,  
 matnr TYPE makt-matnr,  
 spras TYPE makt-spras,  
 maktx TYPE makt-maktx,  
 END OF ty\_makt.  
  
DATA st\_makt TYPE ty\_makt.  
  
Data: it\_makt TYPE TABLE of ty\_makt,  
 wa\_makt LIKE LINE OF it\_makt. *"It creates the internal table perspective can store one row internal table can store multiple rooms*  
  
select matnr spras maktx from makt INTO TABLE it\_makt.  
 Loop at it\_makt INTO wa\_makt. *"Here instead of select statement we can use Loop statement*  
  
 Write: wa\_makt-matnr,  
 wa\_makt-spras,  
 wa\_makt-maktx.  
 ENDLOOP.

If you just need data one time, you can do select and select that is better. But if you need that data more than once in your program, then use internal tables.

Example 1:-

Write a program to print makt fields with the help of Internal Table.

PARAMETERS p\_matnr TYPE makt-matnr.  
  
TYPES: BEGIN OF ty\_makt,  
 matnr TYPE makt-matnr,  
 spras TYPE makt-spras,  
 maktx TYPE makt-maktx,  
 END OF ty\_makt.  
  
DATA: it\_makt Type TABLE of ty\_makt,  
 wa\_makt LIKE LINE OF it\_makt.  
  
Select matnr spras maktx FROM makt INTO TABLE it\_makt.  
 Loop at it\_makt INTO wa\_makt.  
  
 WRITE:/ wa\_makt-matnr,  
 wa\_makt-spras,  
 wa\_makt-maktx.  
  
 ENDLOOP.

Example: 2

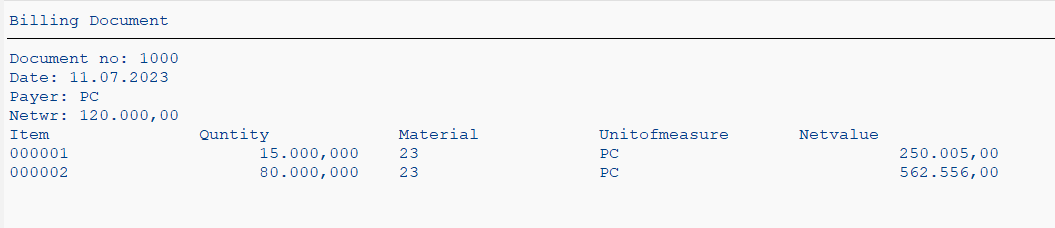
Write a code for vbrk and vbrp using internal table.

PARAMETERS p\_vbeln TYPE zmahe\_first01-vbeln.  
  
  
TYPES: BEGIN OF ty\_vbrk,  
 vbeln TYPE zmahe\_first01-vbeln,  
 fkdat TYPE zmahe\_first01-fkdat,  
 kunrg TYPE zmahe\_first01-kunrg,  
 netwr TYPE zmahe\_first01-netwr,  
 END OF ty\_vbrk.  
  
TYPES: BEGIN OF ty\_vbrp,  
 posnr TYPE zmahe\_first02-posnr,  
 fkimg TYPE zmahe\_first02-fkimg,  
 matnr TYPE zmahe\_first02-matnr,  
 vrkme TYPE zmahe\_first02-vrkme,  
 netwr TYPE zmahe\_first02-netwr,  
 END OF ty\_vbrp.  
  
DATA:it\_vbrk TYPE TABLE OF ty\_vbrk,  
 wa\_vbrk LIKE LINE OF it\_vbrk,  
 it\_vbrp TYPE TABLE OF ty\_vbrp,  
 wa\_vbrp LIKE LINE OF it\_vbrp.  
  
DATA v\_name TYPE kna1-name1.  
DATA v\_mseht TYPE t006a-mseht.  
  
SELECT vbeln fkdat kunrg netwr FROM zmahe\_first01 INTO TABLE it\_vbrk WHERE vbeln EQ p\_vbeln.  
SELECT SINGLE name1 FROM kna1 INTO v\_name WHERE name1 EQ wa\_vbrk-kunrg.  
LOOP AT it\_vbrk INTO wa\_vbrk.  
 WRITE:/ 'Document no:', wa\_vbrk-vbeln,  
 / 'Date:', wa\_vbrk-fkdat,  
 / 'Payer:', wa\_vbrk-kunrg,  
 / 'Netwr:', wa\_vbrk-netwr LEFT-JUSTIFIED.  
  
ENDLOOP.  
  
WRITE:/ 'Item',   
20 'Quantity',  
40 'Material',  
60 'Unitofmeasure',  
80 'Net value'.  
  
SELECT posnr fkimg matnr vrkme netwr FROM zmahe\_first02 INTO TABLE it\_vbrp WHERE vbeln eq p\_vbeln.  
 SELECT SINGLE mseht FROM t006a INTO v\_mseht WHERE mseht eq wa\_vbrp-vrkme.  
 LOOP at it\_vbrp INTO wa\_vbrp.  
 WRITE:/ wa\_vbrp-posnr,  
 20 wa\_vbrp-fkimg,  
 40 wa\_vbrp-matnr,  
 60 wa\_vbrp-vrkme,  
 80 wa\_vbrp-netwr.  
  
 ENDLOOP.

………………………………………………………………………………………………

Example 3:- Here we are using internal table in the Read statement

PARAMETERS p\_vbeln TYPE zmahe\_first01-vbeln.  
  
  
TYPES: BEGIN OF ty\_vbrk,  
 vbeln TYPE zmahe\_first01-vbeln,  
 fkdat TYPE zmahe\_first01-fkdat,  
 kunrg TYPE zmahe\_first01-kunrg,  
 netwr TYPE zmahe\_first01-netwr,  
 END OF ty\_vbrk.  
  
TYPES: BEGIN OF ty\_vbrp,  
 posnr TYPE zmahe\_first02-posnr,  
 fkimg TYPE zmahe\_first02-fkimg,  
 matnr TYPE zmahe\_first02-matnr,  
 vrkme TYPE zmahe\_first02-vrkme,  
 netwr TYPE zmahe\_first02-netwr,  
 END OF ty\_vbrp.  
  
DATA:it\_vbrk TYPE TABLE OF ty\_vbrk,  
 wa\_vbrk LIKE LINE OF it\_vbrk,  
 it\_vbrp TYPE TABLE OF ty\_vbrp,  
 wa\_vbrp LIKE LINE OF it\_vbrp.  
  
TYPES: BEGIN OF ty\_kna1,  
 name1 TYPE kna1-name1,  
 END OF ty\_kna1.  
  
TYPES: BEGIN OF ty\_t006a,  
 mseht TYPE t006a-mseht, *"Here we taking the structure type*  
 END OF ty\_t006a.  
  
  
  
DATA: it\_kna1 TYPE TABLE OF ty\_kna1,  
 wa\_kna1 LIKE LINE OF it\_kna1. *"Converting the structure type to internal type.*  
DATA:it\_t006a TYPE TABLE OF ty\_t006a,  
 wa\_t006a LIKE LINE OF it\_t006a.  
  
SELECT vbeln fkdat kunrg netwr FROM zmahe\_first01 INTO TABLE it\_vbrk WHERE vbeln EQ p\_vbeln.  
LOOP AT it\_vbrk INTO wa\_vbrk. *"Here we sending the data from internal table to work area it contain same table fields*  
 WRITE: / 'Document no:',wa\_vbrk-vbeln,  
 / 'Date:',wa\_vbrk-fkdat,  
 / 'Payer:',wa\_vbrk-kunrg,  
 / 'Netwr:',wa\_vbrk-netwr LEFT-JUSTIFIED.  
  
ENDLOOP.  
  
WRITE: / 'Item',   
20 'Quantity',   
40 'Material',   
60 'Unitofmeasure',   
80 'Net value'.  
  
SELECT posnr fkimg matnr vrkme netwr FROM zmahe\_first02 INTO TABLE it\_vbrp WHERE vbeln EQ p\_vbeln.  
LOOP AT it\_vbrp INTO wa\_vbrp.  
 READ TABLE it\_kna1 INTO wa\_kna1 WITH KEY name1 = wa\_vbrk-kunrg. *"Here we comparing the data from work area in Read table*  
 READ TABLE it\_t006a INTO wa\_t006a WITH KEY mseht = wa\_vbrp-vrkme.  
 WRITE: / wa\_vbrp-posnr,  
 20 wa\_vbrp-fkimg,  
 40 wa\_vbrp-matnr,  
 60 wa\_vbrp-vrkme,  
 80 wa\_vbrp-netwr.  
  
ENDLOOP.



Example 3: **Write a program to show Material number ,plant number, storage location and material description plant name using plant number using internal table**

PARAMETERS r\_werks TYPE mard-werks.  
  
  
  
  
TYPES: BEGIN OF ty\_mard,  
 matnr TYPE mard-matnr,  
 werks TYPE mard-werks,  
 lgort TYPE mard-lgort,  
 END OF ty\_mard.  
  
TYPES: BEGIN OF ty\_makt,  
 matnr TYPE makt-matnr,  
 maktx TYPE makt-maktx,  
 END OF ty\_makt.  
  
DATA:it\_mard TYPE TABLE OF ty\_mard,  
 wa\_mard LIKE LINE OF it\_mard.  
DATA:it\_makt TYPE TABLE OF ty\_makt,  
 wa\_makt LIKE LINE OF it\_makt.  
  
DATA:v\_name TYPE t001w-name1.  
  
SELECT matnr werks lgort FROM mard INTO TABLE it\_mard WHERE werks EQ r\_werks.  
SELECT matnr maktx FROM makt INTO TABLE it\_makt FOR ALL ENTRIES IN it\_mard WHERE matnr EQ it\_mard-matnr AND spras EQ sy-langu.  
SELECT SINGLE name1 FROM t001w INTO v\_name WHERE werks EQ r\_werks.  
LOOP AT it\_mard INTO wa\_mard.  
 READ TABLE it\_makt INTO wa\_makt WITH KEY matnr = wa\_mard-matnr.  
 WRITE:/ wa\_mard-matnr,  
 wa\_mard-werks,  
 wa\_mard-lgort,  
 wa\_makt-maktx,  
 v\_name.  
  
ENDLOOP.



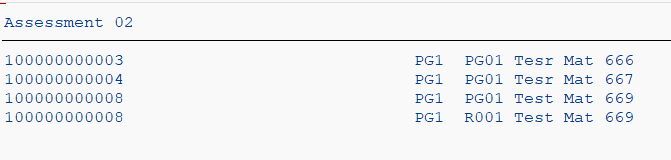
Select-options

DATA p\_matnr TYPE makt-matnr. *"Here we refer the makt-matnr value to p\_matnr*  
SELECT-OPTIONS v\_matnr FOR p\_matnr. *"Here we using select options to discover the range*

data it\_makt TYPE TABLE of makt.  
DATA s\_matnr TYPE makt-matnr. *"Here we refer the makt-matnr value to s\_matnr*  
SELECT-OPTIONS v\_matnr FOR s\_matnr. *"Here we using select options to discover the range*  
  
SELECT \* FROM makt INTO TABLE it\_makt WHERE matnr in v\_matnr [].  
  
V\_matnr-sign = 'I'.  
V\_matnr-option = 'EQ'.  
V\_matnr-low = '25'.  
V\_matnr-high = '40'. *"This are the ranges for field*  
APPEND v\_matnr.  
  
Clear v\_matnr. *"Structure*  
CLEAR v\_matnr []. *"Table*

**Example 1:- Write a program using select option to execute the range .**

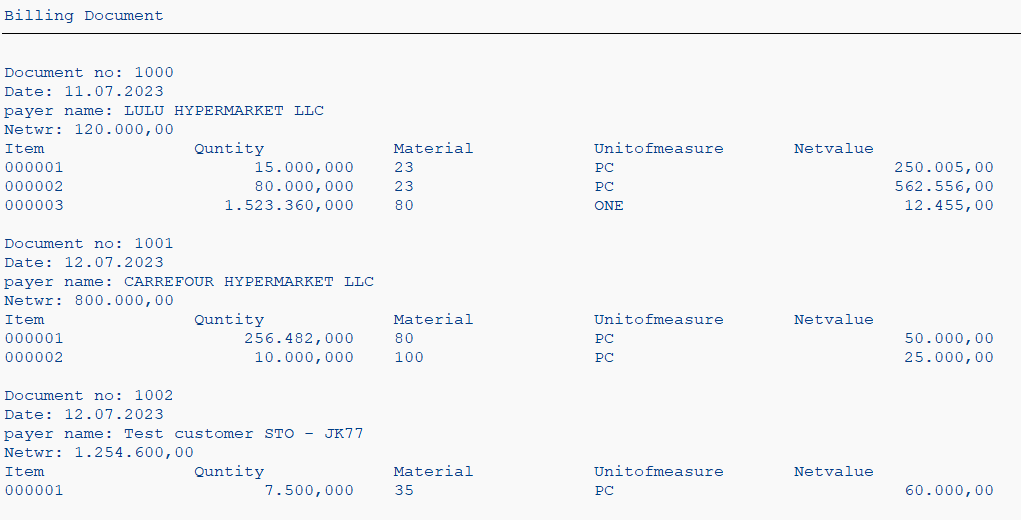
data p\_werks TYPE mard-werks.  
SELECT-OPTIONS s\_werks for p\_werks.  
  
  
  
  
TYPES: BEGIN OF ty\_mard,  
 matnr TYPE mard-matnr,  
 werks TYPE mard-werks,  
 lgort TYPE mard-lgort,  
 END OF ty\_mard.  
  
TYPES: BEGIN OF ty\_makt,  
 matnr TYPE makt-matnr,  
 maktx TYPE makt-maktx,  
 END OF ty\_makt.  
  
DATA:it\_mard TYPE TABLE OF ty\_mard,  
 wa\_mard LIKE LINE OF it\_mard.  
DATA: it\_makt TYPE TABLE OF ty\_makt,  
 wa\_makt LIKE LINE OF it\_makt.  
  
TYPES: BEGIN OF ty\_t001w,  
 name1 TYPE t001w-name1,  
 werks TYPE t001w-werks,  
 END OF ty\_t001w.  
  
 Data:it\_t001w TYPE TABLE of ty\_t001w,  
 wa\_t001w LIKE LINE OF it\_t001w.  
  
SELECT matnr werks lgort FROM mard INTO TABLE it\_mard WHERE werks in s\_werks[].  
SELECT matnr maktx FROM makt INTO TABLE it\_makt FOR ALL ENTRIES IN it\_mard WHERE matnr EQ it\_mard-matnr AND spras EQ sy-langu.  
SELECT werks name1 FROM t001w INTO TABLE it\_t001w FOR ALL ENTRIES IN it\_mard WHERE werks eq it\_mard-werks.  
LOOP AT it\_mard INTO wa\_mard.  
 READ TABLE it\_makt INTO wa\_makt WITH KEY matnr = wa\_mard-matnr.  
 Read TABLE it\_t001w INTO wa\_t001w with key werks = wa\_mard-werks.  
 WRITE:/ wa\_mard-matnr,  
 wa\_mard-werks,  
 wa\_mard-lgort,  
 wa\_makt-maktx,  
 wa\_t001w-name1.  
  
  
ENDLOOP.



……………………………………………………………………………………………………………………………………………………………………………………………………

Example 2:- Write a program to display the document and its iteam below that document details

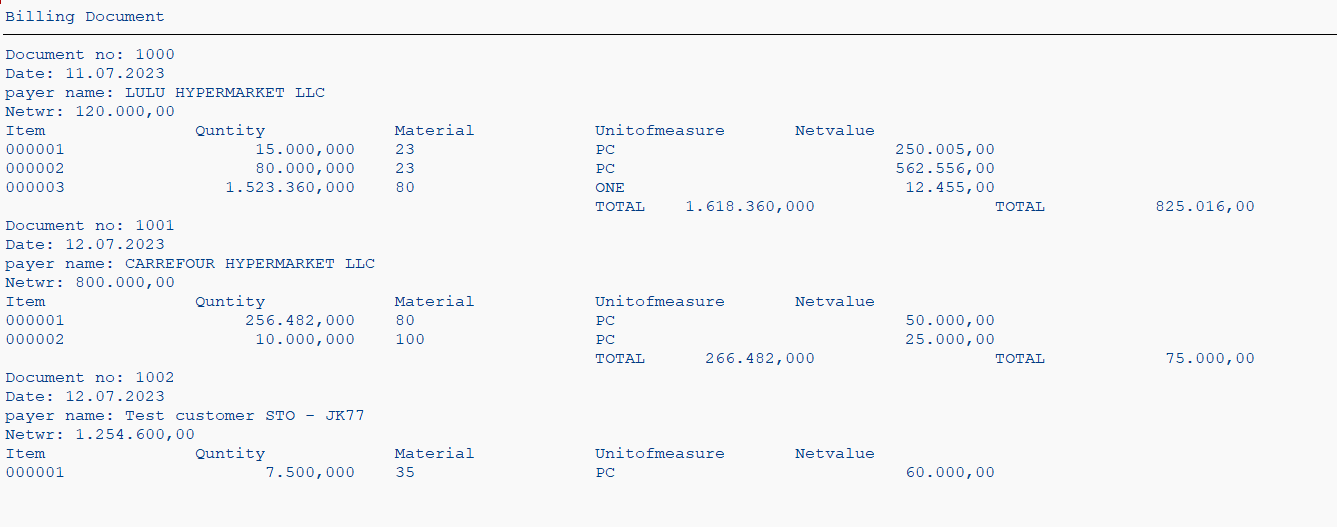
DATA p\_vbeln TYPE zmahe\_first01-vbeln.  
SELECT-OPTIONS v\_vbeln FOR p\_vbeln.  
  
  
TYPES: BEGIN OF ty\_vbrk,  
 vbeln TYPE zmahe\_first01-vbeln,  
 fkdat TYPE zmahe\_first01-fkdat,  
 kunrg TYPE zmahe\_first01-kunrg,  
 netwr TYPE zmahe\_first01-netwr,  
 END OF ty\_vbrk.  
  
TYPES: BEGIN OF ty\_vbrp,  
 vbeln TYPE zmahe\_first02-vbeln,  
 posnr TYPE zmahe\_first02-posnr,  
 fkimg TYPE zmahe\_first02-fkimg,  
 matnr TYPE zmahe\_first02-matnr,  
 vrkme TYPE zmahe\_first02-vrkme,  
 netwr TYPE zmahe\_first02-netwr,  
 END OF ty\_vbrp.  
  
DATA:it\_vbrk TYPE TABLE OF ty\_vbrk,  
 wa\_vbrk LIKE LINE OF it\_vbrk,  
 it\_vbrp TYPE TABLE OF ty\_vbrp,  
 wa\_vbrp LIKE LINE OF it\_vbrp.  
  
TYPES: BEGIN OF ty\_kna1,  
 kunnr TYPE kna1-kunnr,  
 name1 TYPE kna1-name1,  
 END OF ty\_kna1.  
  
TYPES: BEGIN OF ty\_t006a,  
 mseht TYPE t006a-mseht,  
 msehi TYPE t006a-msehi, *"Here we taking the structure type*  
 END OF ty\_t006a.  
  
  
  
DATA:it\_kna1 TYPE TABLE OF ty\_kna1,  
 wa\_kna1 LIKE LINE OF it\_kna1. *"Converting the structure type to internal type.*  
DATA:it\_t006a TYPE TABLE OF ty\_t006a,  
 wa\_t006a LIKE LINE OF it\_t006a.  
  
SELECT vbeln fkdat kunrg netwr FROM zmahe\_first01 INTO TABLE it\_vbrk WHERE vbeln IN v\_vbeln[].  
SELECT vbeln posnr fkimg matnr vrkme netwr FROM zmahe\_first02 INTO TABLE it\_vbrp WHERE vbeln IN v\_vbeln[].  
SELECT kunnr name1 FROM kna1 INTO TABLE it\_kna1 FOR ALL ENTRIES IN it\_vbrk WHERE kunnr EQ it\_vbrk-kunrg.  
  
SORT it\_vbrp BY vbeln.  
DATA v\_vbeln\_temp LIKE wa\_vbrp-vbeln.  
LOOP AT it\_vbrp INTO wa\_vbrp.  
 READ TABLE it\_vbrk INTO wa\_vbrk WITH KEY vbeln = wa\_vbrp-vbeln.  
 READ TABLE it\_kna1 INTO wa\_kna1 WITH KEY kunnr = wa\_vbrk-kunrg. *"Here we comparing the data from work area in Read table*  
 READ TABLE it\_t006a INTO wa\_t006a WITH KEY msehi = wa\_vbrp-vrkme.  
  
 IF v\_vbeln\_temp NE wa\_vbrp-vbeln.  
 WRITE: / 'Document no:’ wa\_vbrk-vbeln,  
 / ‘Date:’ wa\_vbrk-fkdat,  
 / 'payer name:’ wa\_kna1-name1,  
 / ‘Netwr:’ wa\_vbrk-netwr LEFT-JUSTIFIED.  
  
  
 WRITE: / 'Item',  
 20 'Quantity',  
 40 'Material',  
 60 'Unitofmeasure',  
 80 'Net value'.  
  
 ENDIF.  
  
  
  
 WRITE: / wa\_vbrp-posnr,  
 20 wa\_vbrp-fkimg,  
 40 wa\_vbrp-matnr,  
 60 wa\_vbrp-vrkme,  
 80 wa\_vbrp-netwr.  
  
 v\_vbeln\_temp = wa\_vbrp-vbeln.  
  
ENDLOOP.



………………………………………………………………………………………………………………………………………………………………………………………………………

**Example 3: Write a program to calculate the net value in items.**

DATA p\_vbeln TYPE zmahe\_first01-vbeln.  
SELECT-OPTIONS v\_vbeln FOR p\_vbeln.  
  
  
TYPES: BEGIN OF ty\_vbrk,  
 vbeln TYPE zmahe\_first01-vbeln,  
 fkdat TYPE zmahe\_first01-fkdat,  
 kunrg TYPE zmahe\_first01-kunrg,  
 netwr TYPE zmahe\_first01-netwr,  
 END OF ty\_vbrk.  
  
TYPES: BEGIN OF ty\_vbrp,  
 vbeln TYPE zmahe\_first02-vbeln,  
 posnr TYPE zmahe\_first02-posnr,  
 fkimg TYPE zmahe\_first02-fkimg,  
 matnr TYPE zmahe\_first02-matnr,  
 vrkme TYPE zmahe\_first02-vrkme,  
 netwr TYPE zmahe\_first02-netwr,  
 END OF ty\_vbrp.  
  
DATA:it\_vbrk TYPE TABLE OF ty\_vbrk,  
 wa\_vbrk LIKE LINE OF it\_vbrk,  
 it\_vbrp TYPE TABLE OF ty\_vbrp,  
 wa\_vbrp LIKE LINE OF it\_vbrp.  
  
TYPES: BEGIN OF ty\_kna1,  
 kunnr TYPE kna1-kunnr,  
 name1 TYPE kna1-name1,  
 END OF ty\_kna1.  
  
TYPES: BEGIN OF ty\_t006a,  
 mseht TYPE t006a-mseht,  
 msehi TYPE t006a-msehi, *"Here we taking the structure type*  
 END OF ty\_t006a.  
  
  
  
DATA:it\_kna1 TYPE TABLE OF ty\_kna1,  
 wa\_kna1 LIKE LINE OF it\_kna1. *"Converting the structure type to internal type.*  
DATA:it\_t006a TYPE TABLE OF ty\_t006a,  
 wa\_t006a LIKE LINE OF it\_t006a.  
  
  
DATA :v\_total TYPE zmahe\_first02-fkimg VALUE 0 .  
DATA :v\_totalNetV TYPE zmahe\_first02-netwr VALUE 0 .  
DATA: my\_flag TYPE boolean.  
DATA :counter TYPE i .  
  
SELECT vbeln fkdat kunrg netwr FROM zmahe\_first01 INTO TABLE it\_vbrk WHERE vbeln IN v\_vbeln[].  
SELECT vbeln posnr fkimg matnr vrkme netwr FROM zmahe\_first02 INTO TABLE it\_vbrp WHERE vbeln IN v\_vbeln[].  
SELECT kunnr name1 FROM kna1 INTO TABLE it\_kna1 FOR ALL ENTRIES IN it\_vbrk WHERE kunnr EQ it\_vbrk-kunrg.  
  
SORT it\_vbrp BY vbeln.  
DATA v\_vbeln\_temp LIKE wa\_vbrp-vbeln.  
DESCRIBE TABLE it\_vbrp LINES counter.  
LOOP AT it\_vbrp INTO wa\_vbrp.  
 IF v\_vbeln\_temp NE wa\_vbrp-vbeln.  
 IF sy-tabix NE 1 .  
 WRITE : /60 'TOTAL' ,v\_total ,  
 100 'TOTAL',v\_totalNetV .  
 ENDIF .  
 READ TABLE it\_vbrk INTO wa\_vbrk WITH KEY vbeln = wa\_vbrp-vbeln.  
 READ TABLE it\_kna1 INTO wa\_kna1 WITH KEY kunnr = wa\_vbrk-kunrg. *"Here we comparing the data from work area in Read table*  
 READ TABLE it\_t006a INTO wa\_t006a WITH KEY msehi = wa\_vbrp-vrkme.  
  
  
 WRITE:/ 'Document no:',wa\_vbrk-vbeln,  
 / 'Date:',wa\_vbrk-fkdat,  
 / 'payer name:', wa\_kna1-name1,  
 / 'Netwr:',wa\_vbrk-netwr LEFT-JUSTIFIED.  
  
 WRITE:/ 'Item',  
 20 'Quntity',  
 40 'Material',  
 60 'Unitofmeasure',  
 80 'Netvalue'.  
 CLEAR v\_total .  
 CLEAR v\_totalNetV.  
  
 ENDIF.  
 IF wa\_vbrp-vbeln EQ v\_vbeln\_temp OR wa\_vbrp-vbeln IS NOT INITIAL.  
 v\_total = v\_total + wa\_vbrp-fkimg .  
 v\_totalNetV = v\_totalNetV + wa\_vbrp-netwr .  
 ENDIF .  
  
 WRITE:/ wa\_vbrp-posnr,  
 20 wa\_vbrp-fkimg,  
 40 wa\_vbrp-matnr,  
 60 wa\_vbrp-vrkme,  
 80 wa\_vbrp-netwr.  
  
 v\_vbeln\_temp = wa\_vbrp-vbeln.  
 IF SY-TABIX EQ counter.  
 WRITE : /20 'TOTAL' ,v\_total ,  
 80 'TOTAL',v\_totalNetV .  
 ENDIF.  
  
  
ENDLOOP.

****

**…………………………………………………………………**

In internal we use control-Break statements instead of if statement in program.

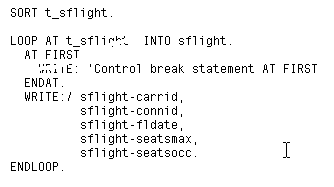
->at the start of the internal table (at first)

->at the end of the internal table (at end, at last).

->when a field value of internal table changes. In order to format & display the Internal Table data according to the above requirement we use

Control Level Statements that helps in reducing the lines of code. Also the Internal Table is to be 'Sorted' before using the Control Level Statements.

1. AT FIRST......ENDAT
2. AT LAST......ENDAT
3. AT NEW <COL>......ENDAT
4. AT END OF <COL>......ENDAT    
   **AT FIRST:**           To display the required data at the beginning of the internal table.



At first

Endat.

At new field name

Endat.

At end of field name

Endat.

**AT** **NEW**.\*If a group of lines of a particular column have same value, it can be displayed at the beginning only once using the control level statement

At New <column name>.

The columns to the left of <column name> are also considered by the 'At New' condition.

Say internal table contents are as follows:

Carrid              connid

AA                  0017                385                  349

AA                  0017                386                  432

AB                   0017                345                 254

AB                  0016                243                  423

"At New Connid."    Works as follows:

 Carrid              connid

   AA                  0017

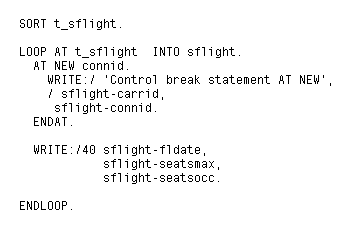
                                               385                  349

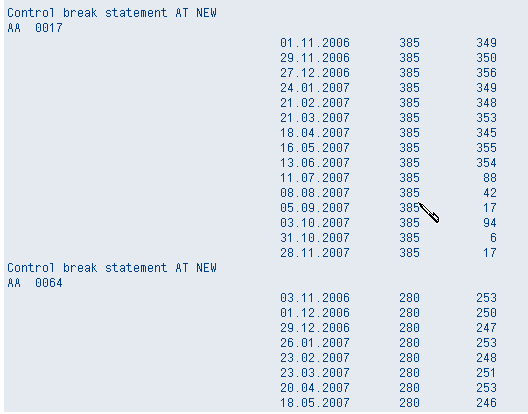
                                               386                 432

   AB                   0017             345                 254

   AB                   0016             243                  423

Ex:

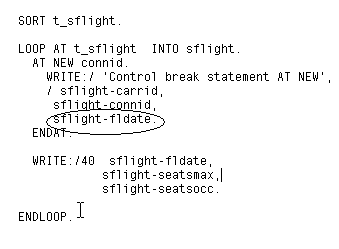


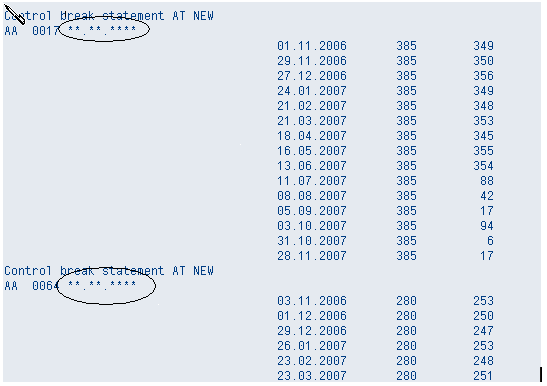
-->The output is displayed as follows:  
   
   
  
 

Ex2:

The columns to the right of the specified column appear as \*\*\*\*\*\* when specified inside the

At New ...Endat loop.

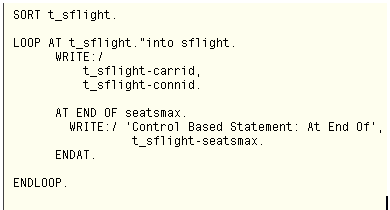


-->The output is displayed as follows:   
   
  
   
   
**AT END OF:** If a group of lines of a particular column have same value, it can be displayed at the End of the group only

  once using the control level statement   At End of  <column name>.

The columns to the left of <column name> are also considered by the 'At End Of ' condition.

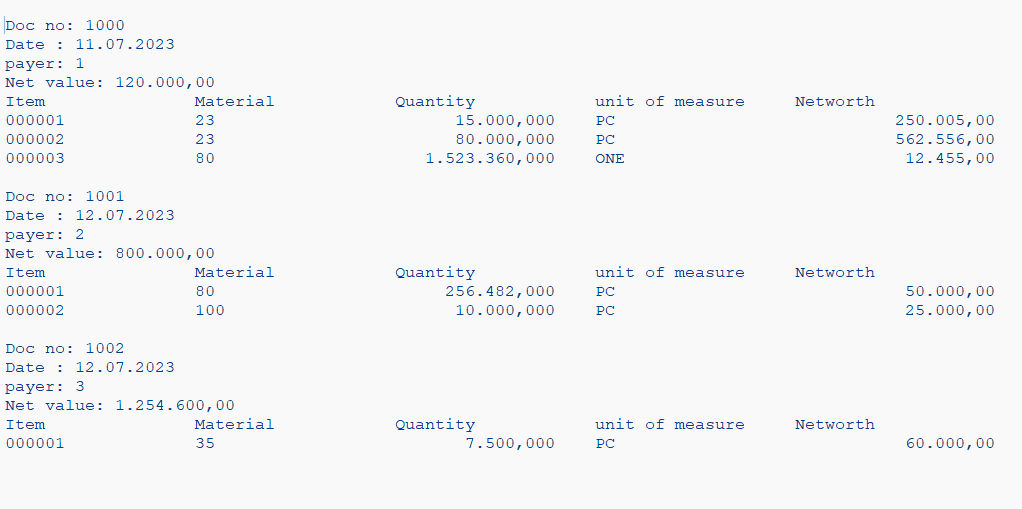
Ex:



Example 1: AT NEW COLUMN

DATA:it\_vbrk TYPE TABLE OF zmahe\_first01,  
 wa\_vbrk LIKE LINE OF it\_vbrk,  
 it\_vbrp TYPE TABLE OF zmahe\_first02,  
 wa\_vbrp LIKE LINE OF it\_vbrp.  
  
*\*SELECTION screen*  
SELECT-OPTIONS v\_vbeln FOR wa\_vbrk-vbeln.  
  
  
*\*Select section*  
SELECT \* FROM zmahe\_first01 INTO TABLE it\_vbrk WHERE vbeln IN v\_vbeln [].  
SELECT \* FROM zmahe\_first02 INTO TABLE it\_vbrp WHERE vbeln IN v\_vbeln [].  
  
*\*Output*

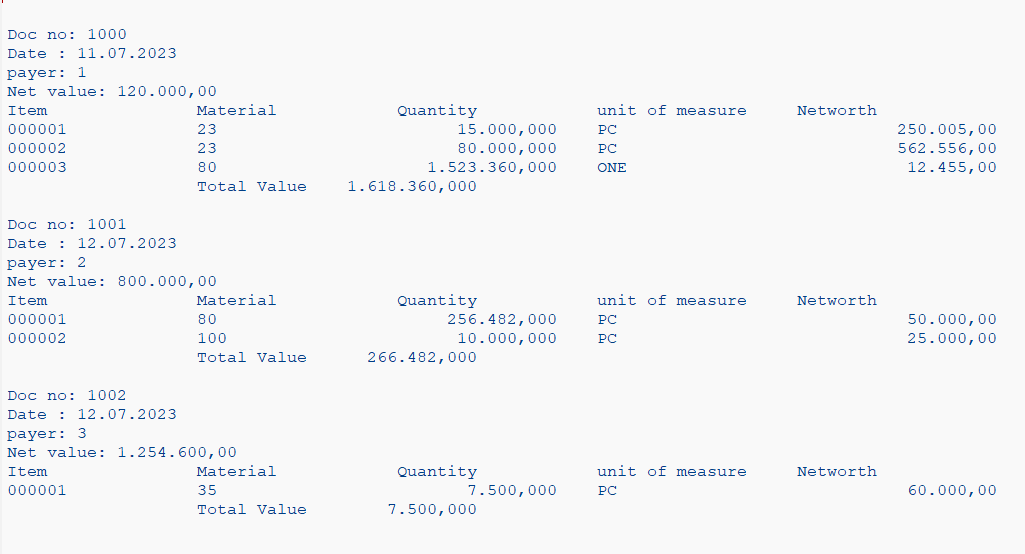
Sort it\_vbrp by vbeln.  
LOOP AT it\_vbrp INTO wa\_vbrp.  
 READ TABLE it\_vbrk INTO wa\_vbrk WITH KEY vbeln = wa\_vbrp-vbeln.  
 AT NEW vbeln. *"Instead of using if statement here we using at new and Endat*  
 skip.  
 WRITE: / 'Doc no:’ wa\_vbrk-vbeln,  
 / ‘Date:’ wa\_vbrk-fkdat,  
 / ‘payer:’ wa\_vbrk-kunrg,  
 / 'Net value:’ wa\_vbrk-netwr LEFT-JUSTIFIED.  
  
 WRITE: / 'Item',  
 20'Material',  
 40'Quantity',  
 60'unit of measure',  
 80'Networth'.  
 ENDAT.  
 WRITE: / wa\_vbrp-posnr,  
 20 wa\_vbrp-matnr,  
 40 wa\_vbrp-fkimg,  
 60 wa\_vbrp-vrkme,  
 80 wa\_vbrp-netwr.  
  
ENDLOOP.



…………………………………………………………………………

Example 2: to calculate the Quantity using control break statements

*\*Data Objects*  
  
DATA:it\_vbrk TYPE TABLE OF zmahe\_first01,  
 wa\_vbrk LIKE LINE OF it\_vbrk,  
 it\_vbrp TYPE TABLE OF zmahe\_first02,  
 wa\_vbrp LIKE LINE OF it\_vbrp.  
  
*\*SELECTION screen*  
SELECT-OPTIONS v\_vbeln FOR wa\_vbrk-vbeln.  
  
  
*\*Select section*  
SELECT \* FROM zmahe\_first01 INTO TABLE it\_vbrk WHERE vbeln IN v\_vbeln[].  
SELECT \* FROM zmahe\_first02 INTO TABLE it\_vbrp WHERE vbeln IN v\_vbeln[].  
  
*\*Output*  
sort it\_vbrp by vbeln.  
LOOP AT it\_vbrp INTO wa\_vbrp.  
 READ TABLE it\_vbrk INTO wa\_vbrk WITH KEY vbeln = wa\_vbrp-vbeln.  
 AT NEW vbeln. *"Instead of using if condition to separate header and item here we using at new and endat*  
 skip.  
 WRITE: / 'Doc no:’ wa\_vbrk-vbeln,  
 / ‘Date:’ wa\_vbrk-fkdat,  
 / ‘payer:’ wa\_vbrk-kunrg,  
 / 'Net value:’ wa\_vbrk-netwr LEFT-JUSTIFIED.  
  
 WRITE: / 'Item',  
 20'Material',  
 40'Quantity',  
 60'unit of measure',  
 80'Networth'.  
 ENDAT.  
 WRITE: / wa\_vbrp-posnr,  
 20 wa\_vbrp-matnr,  
 40 wa\_vbrp-fkimg,  
 60 wa\_vbrp-vrkme,  
 80 wa\_vbrp-netwr.  
  
 at END OF vbeln. *"Here end of is using to calculate the value of quantity*  
 sum.  
 WRITE: /20'Total Value', wa\_vbrp-fkimg.  
 ENDAT.  
  
ENDLOOP.



…………………………………………………………………………

Data Objects

Variables:

The data object in which we can change the data like, first, I can store some value later, change it.

Constants:

*\*Variable, Constants, literals*  
  
Data v\_name TYPE kna1-name1.  
  
v\_name = 'Kiran'.  
v\_name = 'Mahesh'. *"Variables can be modified.*  
  
CONSTANTS c\_date TYPE d VALUE '20230701'.  
  
c\_date = '20230716'. *"Constants can't be modified*

………………………………………………………………………………………………………………………………………………………………………………………………………

TYPES: BEGIN OF ty\_makt,  
 matnr TYPE makt-matnr,  
 spras TYPE makt-spras, *"Here we defining the structure type*  
 maktx TYPE makt-maktx,  
 END OF ty\_makt.  
  
*"TYPES: ty\_t\_makt TYPE TABLE OF ty\_makt. "here we defining structure to table type*  
  
Data ty\_t\_makt TYPE ty\_makt. *"Here its's defining the structure data type*  
  
  
  
DATA st\_makt TYPE ty\_makt. *"here this defining the structure of ty\_makt*

If we refer the data object to a table type. It becomes the internal table.

If we want we can create global data table type.

In some programs there is a restriction in the syntax we cann’t say table of In that case

………………………………………………………………………………………………………………………………………………………………………………………………………

There are three types of Internal Tables.

1. Standard Table
2. Sorted Table
3. Hashed Table
4. Standard Table

Type Standard Table of | Type Table of

Both defines the standard table

TYPES: BEGIN OF ty\_makt,  
 matnr TYPE makt-matnr,  
 spras TYPE makt-spras, *"Here we defining the structure type*  
 maktx TYPE makt-maktx,  
 END OF ty\_makt.  
  
DATA:it\_makt TYPE STANDARD TABLE OF ty\_makt.

This structure have three fields we don’t have to specify anything. All the fields in the standard table are part of them. It has a default key.

Key Definition is not mandatory.

It support non-unique number

Support Index - so we can perform index operation

Perform linear search.

If we say to search the some data it search all the entries.

1. Sorted Table.

Type Sorted Table of

Key definition is mandatory.

It supports both unique and non-unique.

DATA:it\_makt TYPE SORTED TABLE OF ty\_makt WITH UNIQUE KEY matnr.

Entries are always sorted by key.

TYPES: BEGIN OF ty\_makt,  
 matnr TYPE makt-matnr,  
 spras TYPE makt-spras, *"Here we defining the structure type*  
 maktx TYPE makt-maktx,  
 END OF ty\_makt.  
  
data:it\_makt TYPE SORTED TABLE OF ty\_makt WITH UNIQUE KEY matnr spras , *"Here we defining the matnr and spras as key*   
 wa\_makt LIKE LINE OF it\_makt.  
  
wa\_makt-matnr = '100'.  
wa\_makt-spras = 'E'. *"Here matnr and spras are same if we had same table data it ignores*  
wa\_makt-maktx = 'Pen'.  
  
APPEND wa\_makt to it\_makt.  
  
wa\_makt-matnr = '100'.  
wa\_makt-spras = 'D'.  
wa\_makt-maktx = 'Pencil'.  
  
APPEND wa\_makt to it\_makt.

If we mention NON-UNIQUE it allows all the data.

Performs Binary Search.

If we want to search some data its divide the data entries and searches the data.

Support Index - so we can perform index operation

And it checks the number is greater and smaller than the data based on that it search the data.

1. Hashed Table

Type Hashed Table of

It supports only unique key.

Key Definition in Mandatory.

Does not Index operation.

Uses hash function to search for the record.

Read time is constant.

Disadvantage it support the linkage.

Read table it\_makt into wa-makt with key matnr = ‘200’.

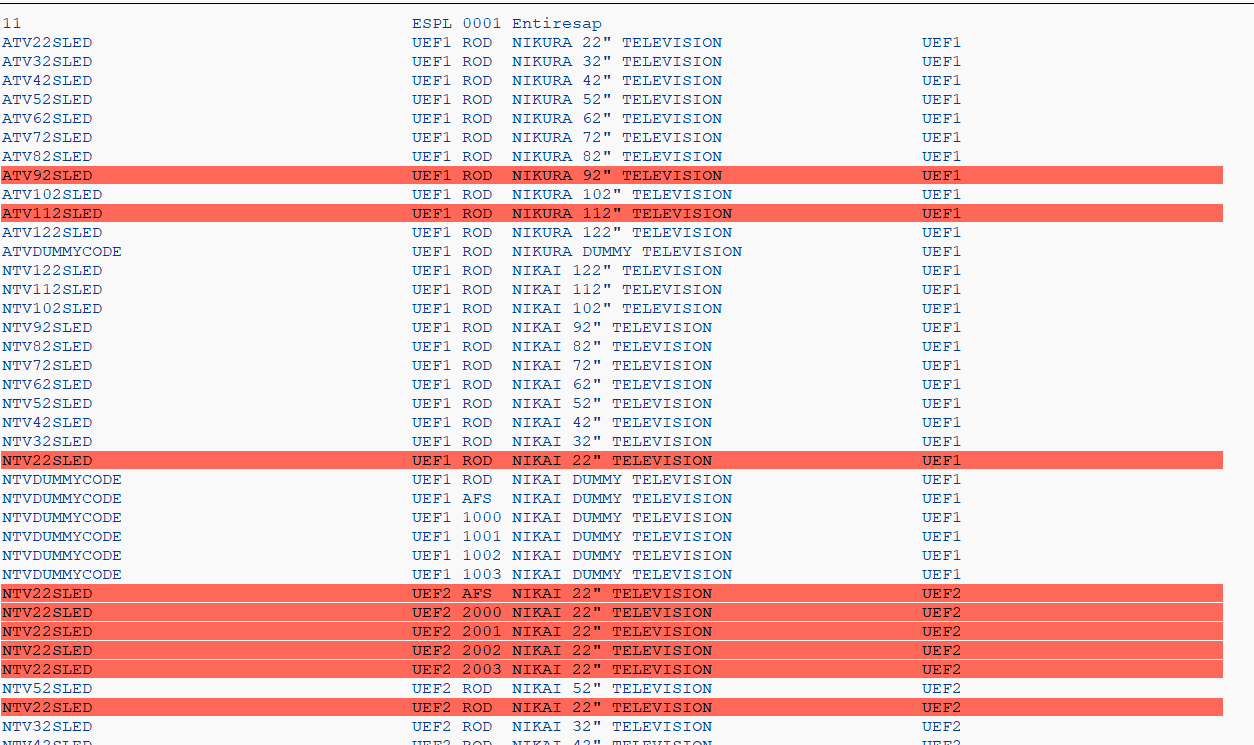
We need to use free key or table key.

If we want ensure the data is unique ‘DELETE ADJACENT DUPLICATES FROM IT\_MAKT COMAPARING MATNR’. If we are okay with duplicates no need to use it.

If we insert or append it forms new row in internal table.

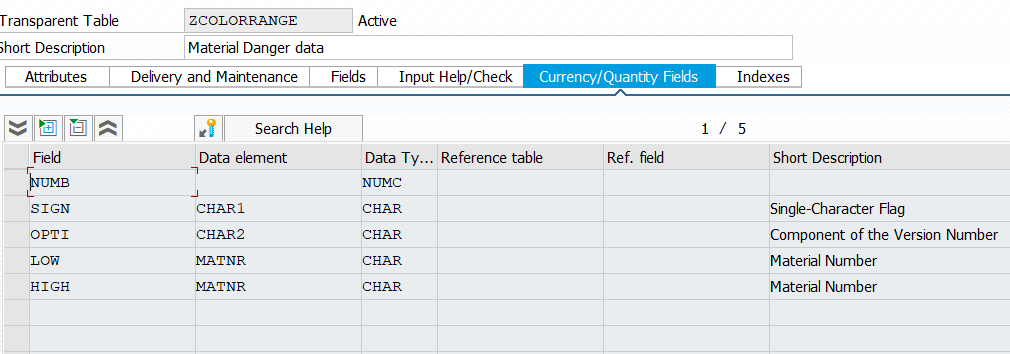
Example 1:- Write a program to give danger to Given materials.

data p\_werks TYPE mard-werks.  
SELECT-OPTIONS s\_werks for p\_werks.  
  
  
  
  
TYPES: BEGIN OF ty\_mard,  
 matnr TYPE mard-matnr,  
 werks TYPE mard-werks,  
 lgort TYPE mard-lgort,  
 END OF ty\_mard.  
  
TYPES: BEGIN OF ty\_makt,  
 matnr TYPE makt-matnr,  
 maktx TYPE makt-maktx,  
 END OF ty\_makt.  
  
DATA:it\_mard TYPE TABLE OF ty\_mard,  
 wa\_mard LIKE LINE OF it\_mard.  
DATA:it\_makt TYPE TABLE OF ty\_makt,  
 wa\_makt LIKE LINE OF it\_makt.  
  
TYPES: BEGIN OF ty\_t001w,  
 name1 TYPE t001w-name1,  
 werks TYPE t001w-werks,  
 END OF ty\_t001w.  
  
 Data:it\_t001w TYPE TABLE of ty\_t001w,  
 wa\_t001w LIKE LINE OF it\_t001w.  
  
SELECT matnr werks lgort FROM mard INTO TABLE it\_mard WHERE werks in s\_werks[].  
SELECT matnr maktx FROM makt INTO TABLE it\_makt FOR ALL ENTRIES IN it\_mard WHERE matnr EQ it\_mard-matnr AND spras EQ sy-langu.  
SELECT werks name1 FROM t001w INTO TABLE it\_t001w FOR ALL ENTRIES IN it\_mard WHERE werks eq it\_mard-werks.  
LOOP AT it\_mard INTO wa\_mard.  
 READ TABLE it\_makt INTO wa\_makt WITH KEY matnr = wa\_mard-matnr.  
 Read TABLE it\_t001w INTO wa\_t001w with key werks = wa\_mard-werks.  
 if wa\_mard-matnr eq 'ATV112SLED' or wa\_mard-matnr eq 'NTV22SLED' or wa\_mard-matnr eq 'ATV92SLED'. *"Here we specify the material name which are danger*  
 FORMAT COLOR COL\_NEGATIVE. *"If the danger we need to specify the key negative if its safe positive.*  
 WRITE:/ wa\_mard-matnr,  
 wa\_mard-werks,  
 wa\_mard-lgort,  
 wa\_makt-maktx,  
 wa\_t001w-name1.  
 FORMAT COLOR OFF. *"After color we need to specify to stop r else it take all materials danger*  
 else.  
 WRITE:/ wa\_mard-matnr,  
 wa\_mard-werks,  
 wa\_mard-lgort,  
 wa\_makt-maktx,  
 wa\_t001w-name1.  
 ENDIF.  
  
  
ENDLOOP.

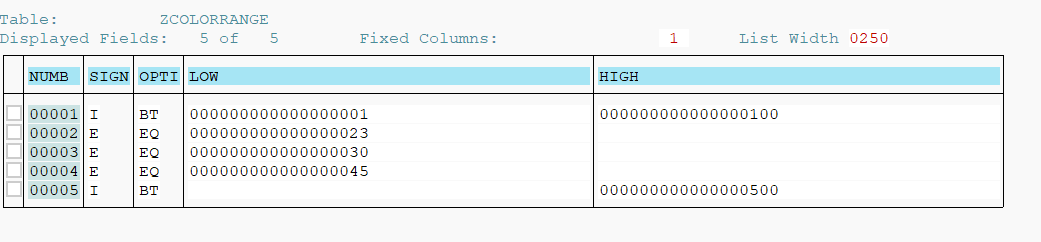


……………………………………………………………………………………

Example 2: Write a program to mark Danger for 25 30 45 in Material.

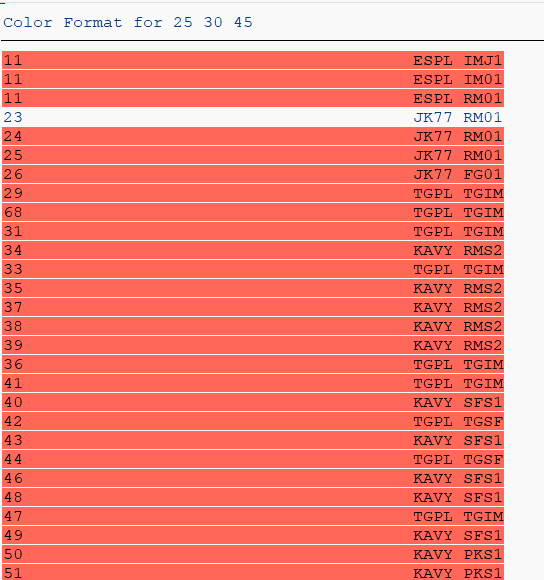


Here we creating a Range table to specify the Ranges of elements.



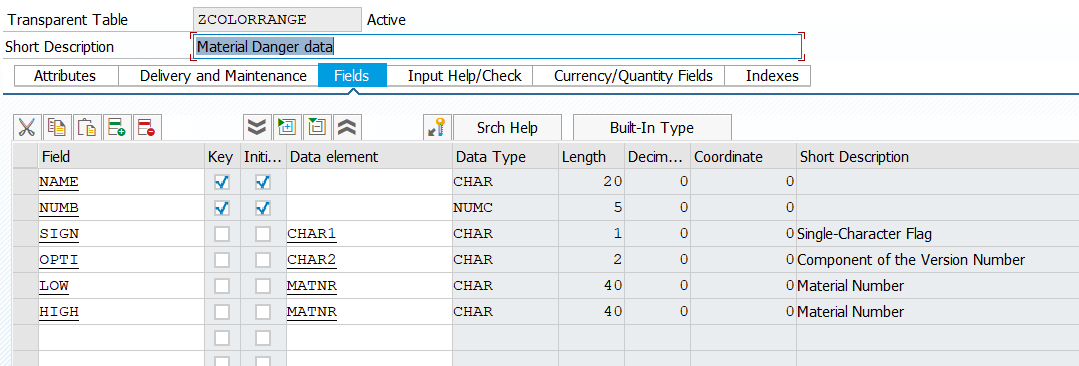
Here we providing the details here we excluding the 23 30 45 and including 1 to 100 and 500

DATA p\_werks TYPE mard-werks.  
SELECT-OPTIONS s\_werks FOR p\_werks.  
  
  
  
  
TYPES: BEGIN OF ty\_mard,  
 matnr TYPE mard-matnr,  
 werks TYPE mard-werks,  
 lgort TYPE mard-lgort,  
 END OF ty\_mard.  
  
  
TYPES: BEGIN OF ty\_range,  
 sign TYPE char1,  
 opti TYPE char2,  
 low TYPE matnr,  
 high TYPE matnr,  
 END OF ty\_range.  
  
  
  
  
DATA:it\_range TYPE TABLE OF ty\_range,  
 wa\_range LIKE LINE OF it\_range.  
  
DATA:it\_mard TYPE TABLE OF ty\_mard,  
 wa\_mard LIKE LINE OF it\_mard.  
  
  
  
SELECT matnr werks lgort FROM mard INTO TABLE it\_mard WHERE werks IN s\_werks[].  
SELECT sign opti low high FROM zcolorrange INTO TABLE it\_range.  
LOOP AT it\_mard INTO wa\_mard.  
 IF wa\_mard-matnr in it\_range. *"here we using "IN" to check matnr in it or not*  
 FORMAT COLOR COL\_NEGATIVE.  
 WRITE:/ wa\_mard-matnr,  
 wa\_mard-werks,  
 wa\_mard-lgort.  
 FORMAT COLOR OFF. *"After color we need to specify to stop r else it take all materials danger*  
 ELSE.  
 WRITE:/ wa\_mard-matnr,  
 wa\_mard-werks,  
 wa\_mard-lgort.  
 ENDIF.  
  
ENDLOOP.



……………………………………………………………………………………

Example 3:- write a program to specify positive and negative numbers



DATA p\_werks TYPE mard-werks.  
SELECT-OPTIONS s\_werks FOR p\_werks.  
  
  
  
  
TYPES: BEGIN OF ty\_mard,  
 matnr TYPE mard-matnr,  
 werks TYPE mard-werks,  
 lgort TYPE mard-lgort,  
 END OF ty\_mard.  
  
  
TYPES: BEGIN OF ty\_range,  
 sign TYPE char1,  
 opti TYPE char2,  
 low TYPE matnr,  
 high TYPE matnr,  
 END OF ty\_range.  
  
  
  
  
DATA:it\_range TYPE TABLE OF zcolorrange,  
 wa\_range LIKE LINE OF it\_range.  
  
DATA:it\_mard TYPE TABLE OF ty\_mard,  
 wa\_mard LIKE LINE OF it\_mard.  
  
DATA: It\_range\_s TYPE STANDARD TABLE OF ty\_range,  
 it\_range\_d TYPE STANDARD TABLE OF ty\_range,  
 wa\_range1 TYPE ty\_range.  
  
  
  
SELECT matnr werks lgort FROM mard INTO TABLE it\_mard WHERE werks IN s\_werks[].  
SELECT \* FROM zcolorrange INTO TABLE it\_range.  
  
  
LOOP AT it\_range INTO wa\_range.  
  
 IF wa\_range-name EQ 'DANGER'.  
 MOVE-CORRESPONDING wa\_range TO wa\_range1.  
 APPEND wa\_range1 TO it\_range\_d.  
 ENDIF.  
 IF wa\_range-name EQ 'SAFE'.  
 MOVE-CORRESPONDING wa\_range TO wa\_range1.  
 APPEND wa\_range1 TO it\_range\_s.  
 ENDIF.  
 CLEAR wa\_range1.  
ENDLOOP.  
  
  
LOOP AT it\_mard INTO wa\_mard.  
  
 IF wa\_mard-matnr IN It\_range\_d.  
  
 FORMAT COLOR COL\_NEGATIVE.  
  
 WRITE:/ wa\_mard-matnr,  
 20 wa\_mard-werks,  
 40 wa\_mard-lgort.  
  
  
 ELSEIF wa\_mard-matnr IN It\_range\_s.  
 FORMAT COLOR COL\_POSITIVE.  
 WRITE:/ wa\_mard-matnr,  
 20 wa\_mard-werks,  
 40 wa\_mard-lgort.  
  
  
 ELSE.  
 WRITE:/ wa\_mard-matnr,  
 wa\_mard-werks,  
 wa\_mard-lgort.  
 ENDIF.  
  
ENDLOOP.

…………………………………………………………………………………………………………………………………………………………………………………………………

MESSAGE

DATA:it\_makt TYPE TABLE OF makt,  
 wa\_makt LIKE LINE OF it\_makt.  
select \* from makt INTO TABLE it\_makt WHERE matnr eq wa\_makt-matnr and spras eq wa\_makt-spras.  
 wa\_makt-matnr = '100'.  
 wa\_makt-spras = 'E'.  
 wa\_makt-maktx = 'Pen'.  
  
 *"insert makt from wa\_makt. "when we try to insert record its shwos record is not exist.*  
  
 if sy-subrc is not INITIAL. *"Afer select statement if select find atleast on record its intial will be 0*  
  
 INSERT makt FROM wa\_makt. *"if we use is initial we check the statement is successful we don't need to insert anything in that*  
  
 else.  
 UPDATE makt FROM wa\_makt.  
  
 ENDIF.

Instead of insert and update we use

If we try to insert record which is already exist it’s get runtime error

DATA:it\_makt TYPE TABLE OF makt,  
 wa\_makt LIKE LINE OF it\_makt.  
select \* from makt INTO TABLE it\_makt WHERE matnr eq wa\_makt-matnr and spras eq wa\_makt-spras.  
 wa\_makt-matnr = '100'.  
 wa\_makt-spras = 'E'.  
 wa\_makt-maktx = 'Pen'.  
  
 *"insert makt from wa\_makt. "when we try to insert record its shwos record is not exist.*  
  
 modify makt FROM wa\_makt.   
 delete makt FROM wa\_makt.

……………………………………………………………………………………………………………………………………………………………………………………………………

Example 1:-

DATA:it\_makt TYPE TABLE OF makt,  
 wa\_makt LIKE LINE OF it\_makt.  
SELECT \* FROM makt INTO TABLE it\_makt WHERE matnr EQ wa\_makt-matnr AND spras EQ wa\_makt-spras.  
wa\_makt-matnr = '100'.  
wa\_makt-spras = 'E'.  
wa\_makt-maktx = 'Pen'.  
  
  
  
*"MODIFY makt FROM wa\_makt.*  
DELETE makt FROM wa\_makt.  
if sy-subrc is INITIAL.  
 MESSAGE 'Record Deleted successfully' TYPE 'I'. *"Here we specify the message and giving information type to that message*  
 ENDIF.

…………………………………………………………………………………………………………………………………………………………………………………………………

E – Error.

I – Information.

S – Status.

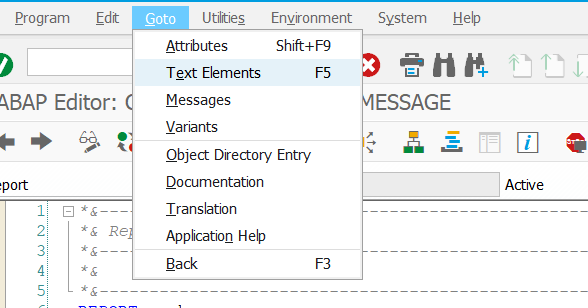
A – Abandon.

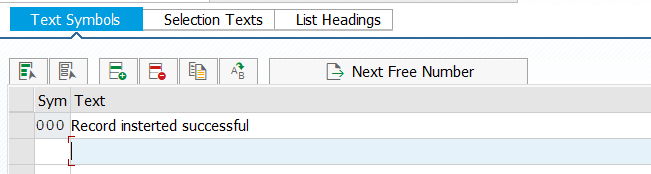
X - short dump.

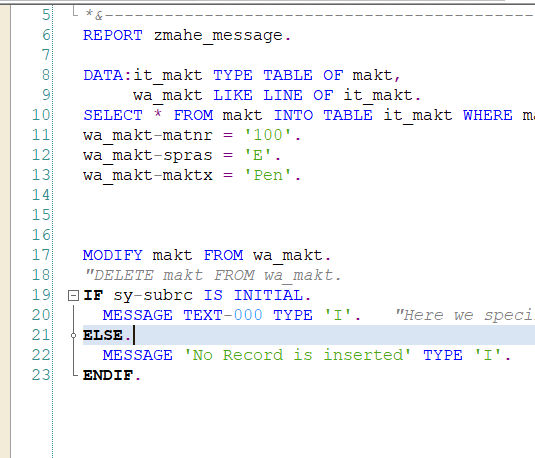
DATA:it\_makt TYPE TABLE OF makt,  
 wa\_makt LIKE LINE OF it\_makt.  
SELECT \* FROM makt INTO TABLE it\_makt WHERE matnr EQ wa\_makt-matnr AND spras EQ wa\_makt-spras.  
wa\_makt-matnr = '100'.  
wa\_makt-spras = 'E'.  
wa\_makt-maktx = 'Pen'.  
  
  
  
*"MODIFY makt FROM wa\_makt.*  
DELETE makt FROM wa\_makt.  
if sy-subrc is INITIAL.  
 MESSAGE 'Record Deleted successfully' TYPE 'I'. *"Here we specify the message and giving information type to that message*  
 else.  
 MESSAGE 'No Record is deleted' TYPE 'I'.  
 ENDIF.

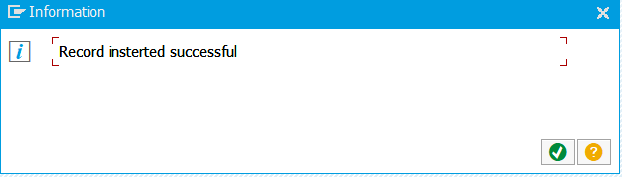


Instead of updating the message in text form we can create text element for that.



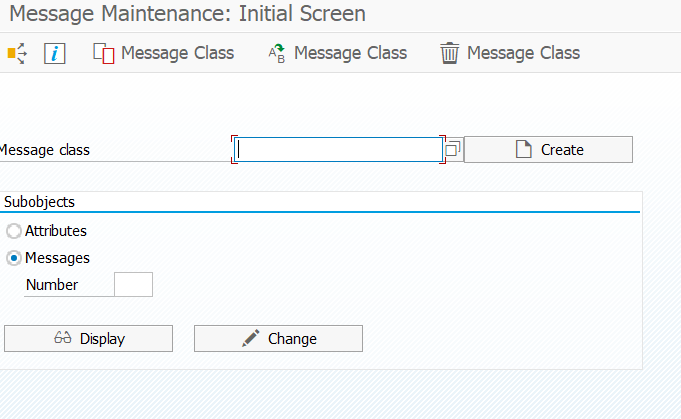


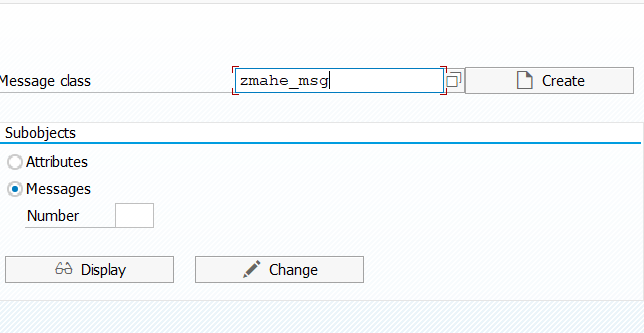


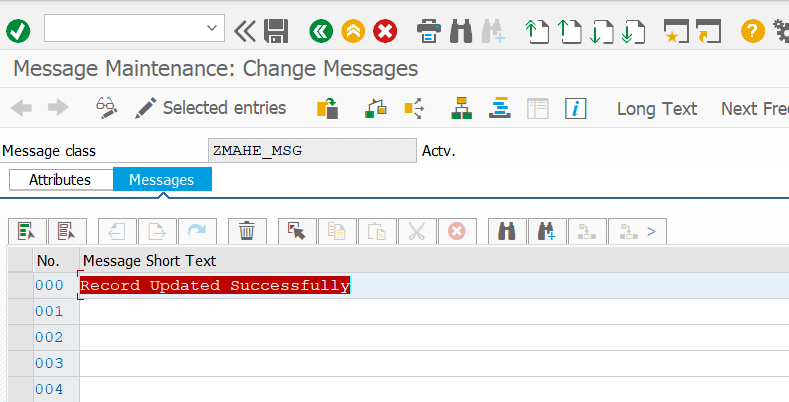


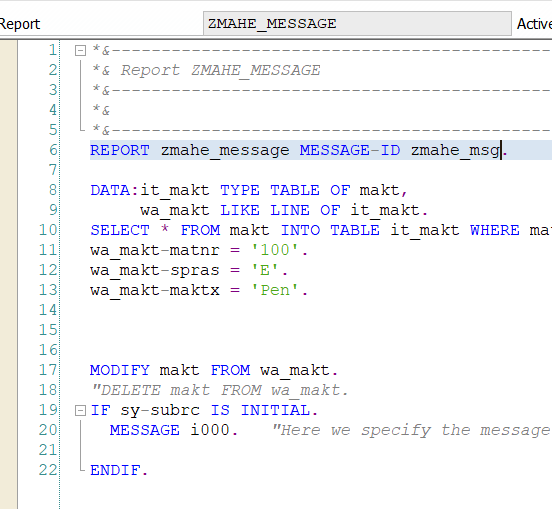
………………………………………………………………………………………………………………………………………………………………………………………………………

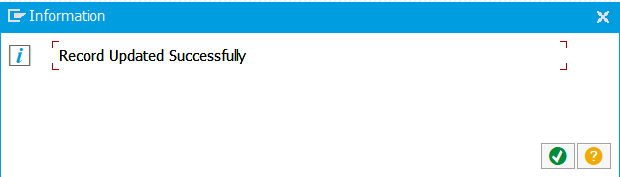
If we want use this message for multiple program we can create in se91







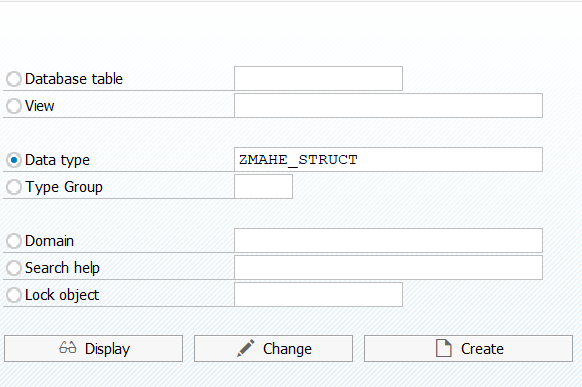


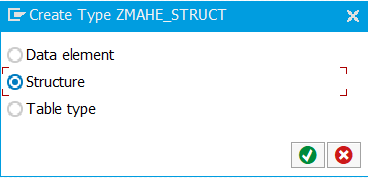


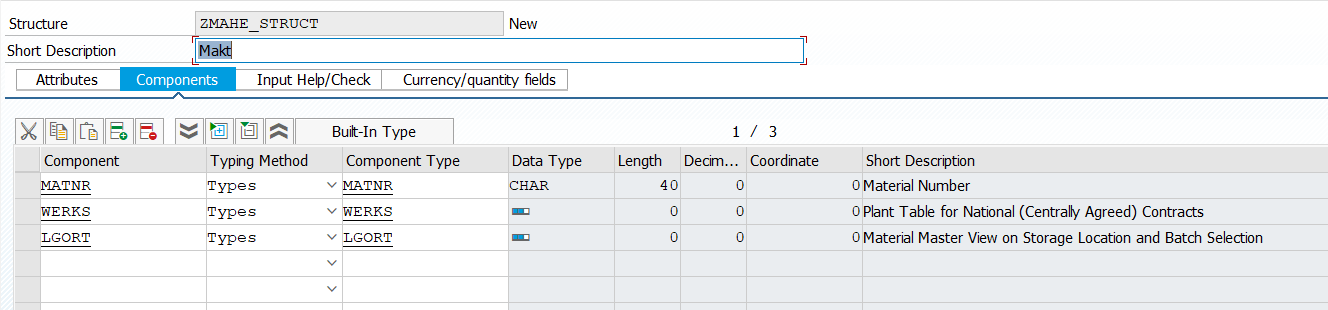
……………………………………………………………………………………………………………………………………………………………………………………………………

If we want a global table type which can be used in multiple program.

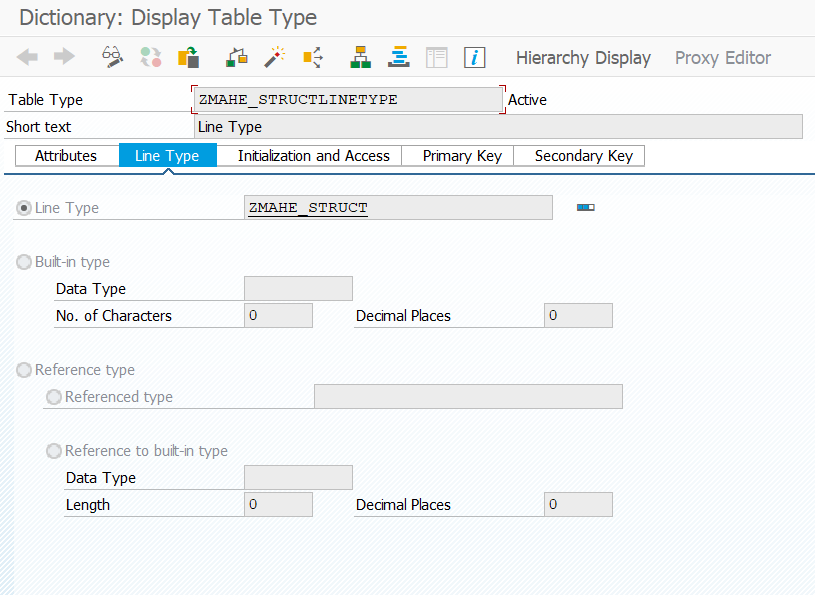
First we need to create structure data type se11







Line Type:



TYPES: BEGIN OF ty\_makt,  
 matnr TYPE makt-matnr,  
 spras TYPE makt-spras,  
 END OF ty\_makt.  
  
TYPES ty\_t\_makt TYPE STANDARD TABLE OF ty\_makt. *"Structure Table Type*  
  
data it\_m TYPE zmahe\_structlinetype. *"here we specify the line type*

…………………………………………………………………………………………………………………………………………………………………………………………………….

1. Program Constructor Event

Load-of-program.

1. Reporting Events(Executable Type Program)

Initialization

Start-of-selection.

End-of-selection.

1. Selection Screen Events

At Selection-Screen OUTPUT.

At Selection-screen

At Selection-Screen ON field.

At Selection-Screen on block.

At Selection-Screen on Help Request.

At Selection-Screen on value Request.

1. List Events

Top-of-page

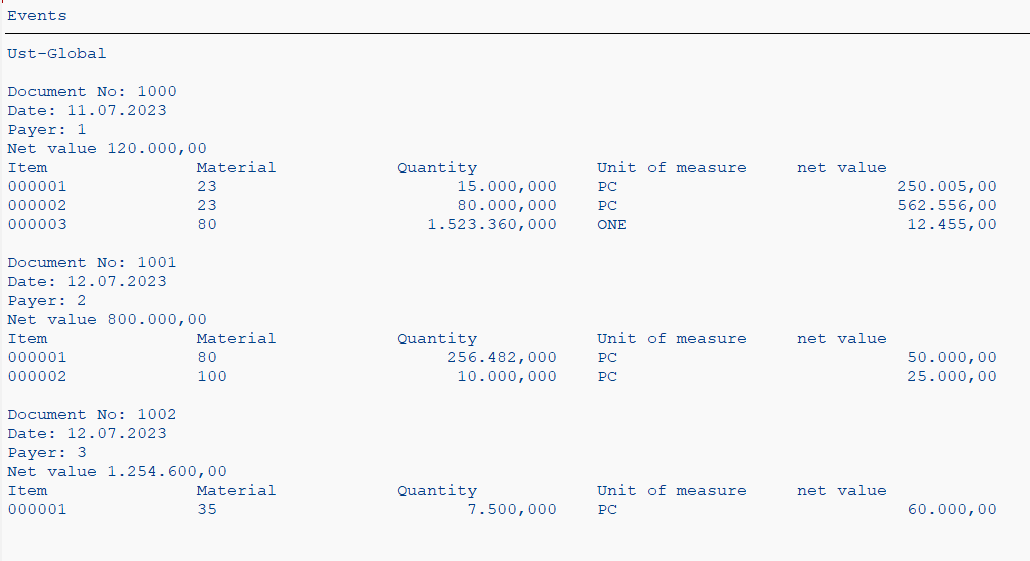
End-of-page

At Line Selection.

At User-Command.

Example 1:-

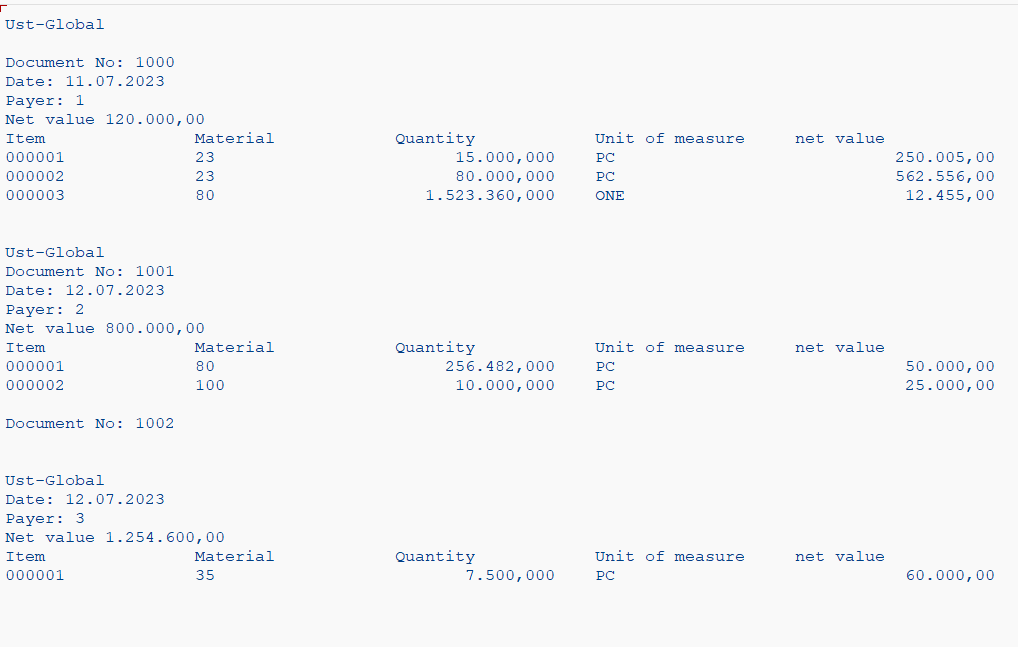
DATA:it\_vbrk TYPE TABLE OF zmahe\_first01,  
 wa\_vbrk LIKE LINE OF it\_vbrk,  
 it\_vbrp TYPE TABLE OF zmahe\_first02,  
 wa\_vbrp LIKE LINE OF it\_vbrp.  
  
SELECT-OPTIONS s\_vbeln FOR wa\_vbrk-vbeln.  
  
START-OF-SELECTION.  
 SELECT \* FROM zmahe\_first01 INTO TABLE it\_vbrk WHERE vbeln IN s\_vbeln[]. *"Here we defining the start of selection for select statements*  
 SELECT \* FROM zmahe\_first02 INTO TABLE it\_vbrp WHERE vbeln IN s\_vbeln[].  
  
END-OF-SELECTION.  
 SORT it\_vbrp BY vbeln.  
 LOOP AT it\_vbrp INTO wa\_vbrp.  
 READ TABLE it\_vbrk INTO wa\_vbrk WITH KEY vbeln = wa\_vbrp-vbeln.  
 AT NEW vbeln.  
 skip.  
 WRITE:/ 'Document No:',wa\_vbrk-vbeln,  
 / 'Date:',wa\_vbrk-fkdat,  
 / 'Payer:',wa\_vbrk-kunrg,  
 / 'Net value',wa\_vbrk-netwr LEFT-JUSTIFIED.  
  
 WRITE:/ 'Item',  
 20 'Material',  
 40 'Quantity',  
 60 'Unit of measure',  
 80 'net value'.  
  
 ENDAT.  
  
 WRITE:/ wa\_vbrp-posnr,  
 20 wa\_vbrp-matnr,  
 40 wa\_vbrp-fkimg,  
 60 wa\_vbrp-vrkme,  
 80 wa\_vbrp-netwr.  
  
 ENDLOOP.  
  
 TOP-OF-PAGE. *"Here Top-of-page is used to name of the company or logo at top the page*  
 WRITE:/ 'Ust-Global'.



………………………………………………………………………………………………………………………………………………………………………………………

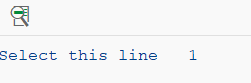
Top of page is called when it encounters the right statement for the first time in your program.

REPORT zmahe\_billing\_events NO STANDARD PAGE HEADING LINE-COUNT 11(1). *"Here we used this to define the top of page on every document*  
  
  
DATA:it\_vbrk TYPE TABLE OF zmahe\_first01,  
 wa\_vbrk LIKE LINE OF it\_vbrk,  
 it\_vbrp TYPE TABLE OF zmahe\_first02,  
 wa\_vbrp LIKE LINE OF it\_vbrp.  
  
SELECT-OPTIONS s\_vbeln FOR wa\_vbrk-vbeln.  
  
START-OF-SELECTION.  
 SELECT \* FROM zmahe\_first01 INTO TABLE it\_vbrk WHERE vbeln IN s\_vbeln[]. *"Here we defining the start of selection for select statements*  
 SELECT \* FROM zmahe\_first02 INTO TABLE it\_vbrp WHERE vbeln IN s\_vbeln[].  
  
END-OF-SELECTION.  
 SORT it\_vbrp BY vbeln.  
 LOOP AT it\_vbrp INTO wa\_vbrp.  
 READ TABLE it\_vbrk INTO wa\_vbrk WITH KEY vbeln = wa\_vbrp-vbeln.  
 AT NEW vbeln.  
 skip.  
 WRITE:/ 'Document No:',wa\_vbrk-vbeln,  
 / 'Date:',wa\_vbrk-fkdat,  
 / 'Payer:',wa\_vbrk-kunrg,  
 / 'Net value',wa\_vbrk-netwr LEFT-JUSTIFIED.  
  
 WRITE:/ 'Item',  
 20 'Material',  
 40 'Quantity',  
 60 'Unit of measure',  
 80 'net value'.  
  
 ENDAT.  
  
 WRITE:/ wa\_vbrp-posnr,  
 20 wa\_vbrp-matnr,  
 40 wa\_vbrp-fkimg,  
 60 wa\_vbrp-vrkme,  
 80 wa\_vbrp-netwr.  
  
 ENDLOOP.  
  
 TOP-OF-PAGE. *"Here Top-of-page is used to name of the company or logo at top the page*  
 WRITE:/ 'Ust-Global'.



Example 2:

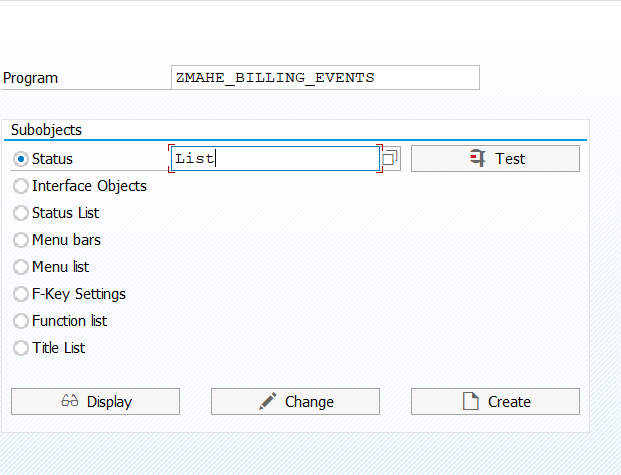
DATA:it\_vbrk TYPE TABLE OF zmahe\_first01,  
 wa\_vbrk LIKE LINE OF it\_vbrk,  
 it\_vbrp TYPE TABLE OF zmahe\_first02,  
 wa\_vbrp LIKE LINE OF it\_vbrp.  
  
SELECT-OPTIONS s\_vbeln FOR wa\_vbrk-vbeln.  
  
START-OF-SELECTION.  
 SELECT \* FROM zmahe\_first01 INTO TABLE it\_vbrk WHERE vbeln IN s\_vbeln[]. *"Here we defining the start of selection for select statements*  
 SELECT \* FROM zmahe\_first02 INTO TABLE it\_vbrp WHERE vbeln IN s\_vbeln[].  
  
END-OF-SELECTION.  
 SORT it\_vbrp BY vbeln.  
 LOOP AT it\_vbrp INTO wa\_vbrp.  
 READ TABLE it\_vbrk INTO wa\_vbrk WITH KEY vbeln = wa\_vbrp-vbeln.  
 AT NEW vbeln.  
 SKIP.  
 WRITE:/ 'Document No:',wa\_vbrk-vbeln,  
 / 'Date:',wa\_vbrk-fkdat,  
 / 'Payer:',wa\_vbrk-kunrg,  
 / 'Net value',wa\_vbrk-netwr LEFT-JUSTIFIED.  
  
 WRITE:/ 'Item',  
 20 'Material',  
 40 'Quantity',  
 60 'Unit of measure',  
 80 'net value'.  
  
 ENDAT.  
  
 WRITE:/ wa\_vbrp-posnr,  
 20 wa\_vbrp-matnr,  
 40 wa\_vbrp-fkimg,  
 60 wa\_vbrp-vrkme,  
 80 wa\_vbrp-netwr.  
  
 ENDLOOP.  
  
TOP-OF-PAGE. *"Here Top-of-page is used to name of the company or logo at top the page*  
 WRITE:/ 'Ust-Global'.  
  
 *"END-OF-PAGE.*  
 *"write:/ 'End Document'.*  
  
AT LINE-SELECTION.  
 WRITE:/ 'Select this line',sy-lsind. *"This is used to select some other information when we tap on the data it takes to select this line*



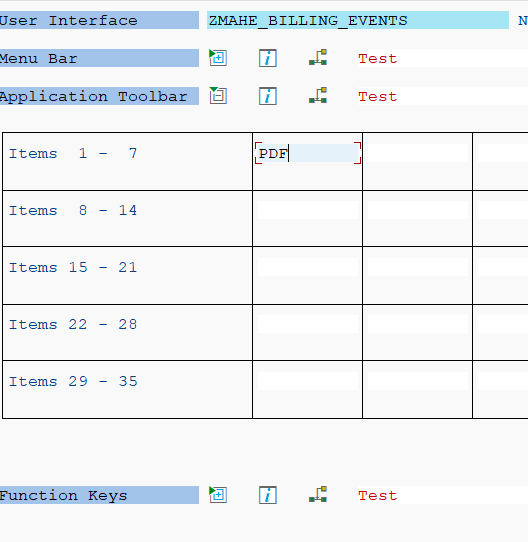
………………………………………………………………………………………

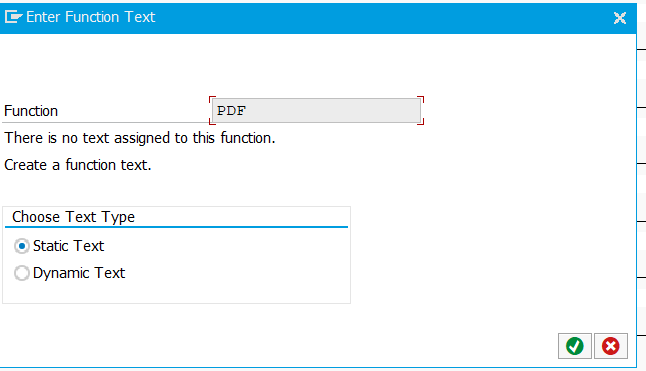
Selection screen is generated when you use parameters or select options. This screen is related when you use the right statement.

To create a user command like pdf and excel se41

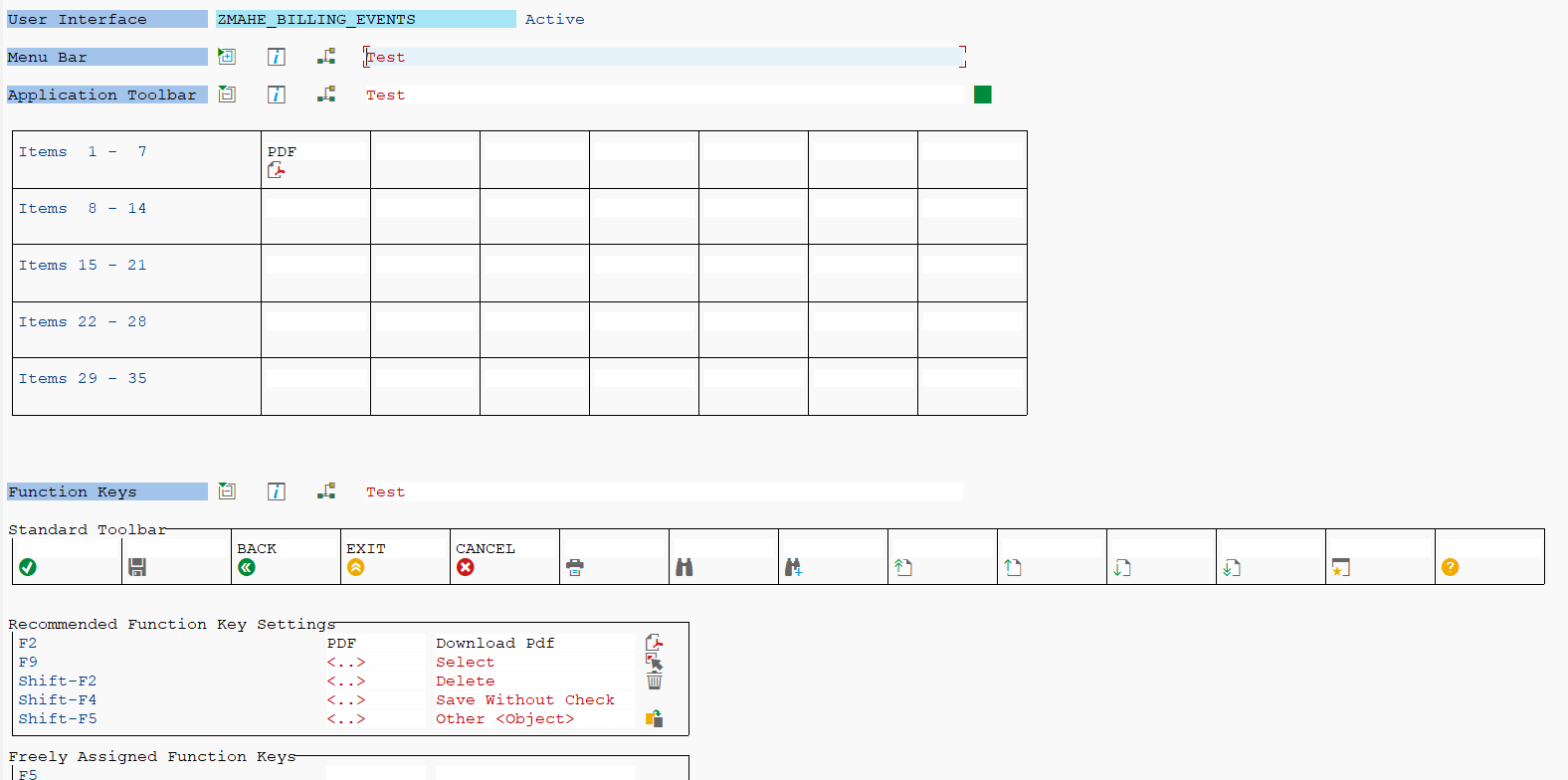








DATA:it\_vbrk TYPE TABLE OF zmahe\_first01,  
 wa\_vbrk LIKE LINE OF it\_vbrk,  
 it\_vbrp TYPE TABLE OF zmahe\_first02,  
 wa\_vbrp LIKE LINE OF it\_vbrp.  
  
SELECT-OPTIONS s\_vbeln FOR wa\_vbrk-vbeln.  
  
START-OF-SELECTION.  
 SELECT \* FROM zmahe\_first01 INTO TABLE it\_vbrk WHERE vbeln IN s\_vbeln[]. *"Here we defining the start of selection for select statements*  
 SELECT \* FROM zmahe\_first02 INTO TABLE it\_vbrp WHERE vbeln IN s\_vbeln[].  
  
END-OF-SELECTION.  
 *"SET PF-STATUS 'List'.*  
 SORT it\_vbrp BY vbeln.  
 LOOP AT it\_vbrp INTO wa\_vbrp.  
 READ TABLE it\_vbrk INTO wa\_vbrk WITH KEY vbeln = wa\_vbrp-vbeln.  
 AT NEW vbeln.  
 SKIP.  
 WRITE:/ 'Document No:',wa\_vbrk-vbeln,  
 / 'Date:',wa\_vbrk-fkdat,  
 / 'Payer:',wa\_vbrk-kunrg,  
 / 'Net value',wa\_vbrk-netwr LEFT-JUSTIFIED.  
  
 WRITE:/ 'Item',  
 20 'Material',  
 40 'Quantity',  
 60 'Unit of measure',  
 80 'net value'.  
  
 ENDAT.  
  
 WRITE:/ wa\_vbrp-posnr,  
 20 wa\_vbrp-matnr,  
 40 wa\_vbrp-fkimg,  
 60 wa\_vbrp-vrkme,  
 80 wa\_vbrp-netwr.  
  
 ENDLOOP.  
  
TOP-OF-PAGE. *"Here Top-of-page is used to name of the company or logo at top the page*  
 WRITE:/ 'Ust-Global'.  
  
 *"END-OF-PAGE.*  
 *"write:/ 'End Document'.*  
  
AT LINE-SELECTION.  
 WRITE:/ 'Select this line',sy-lsind. *"This is used to select some other information when we tap on the data it takes to select this line*  
  
  
AT USER-COMMAND.  
 IF sy-ucomm EQ 'PDF'.  
  
  
 ENDIF.



……………………………………………………………………………………………………………………………………………………………………………………………………

AT USER-COMMAND.  
 case sy-ucomm.  
 when 'PDF'.  
 when 'Excel'.  
   
 ENDCASE.

…………………………………………………………………………………………………………………………………………………………………………………………………….

Example1:- using at line selection.

DATA:it\_vbrk TYPE TABLE OF zmahe\_first01,  
 wa\_vbrk LIKE LINE OF it\_vbrk,  
 it\_vbrp TYPE TABLE OF zmahe\_first02,  
 wa\_vbrp LIKE LINE OF it\_vbrp.  
  
  
START-OF-SELECTION.  
 SELECT \* FROM zmahe\_first01 INTO TABLE it\_vbrk.  
  
  
 SORT it\_vbrp BY vbeln.  
  
 LOOP AT it\_vbrk INTO wa\_vbrk.  
  
 WRITE: 'Document no:', wa\_vbrk-vbeln,  
 / 'Date:', wa\_vbrk-fkdat,  
 / 'payer:', wa\_vbrk-kunrg,  
 / 'net value',wa\_vbrk-netwr.  
 HIDE wa\_vbrk-vbeln. *"if u want the system to remember what ur print in the line we use hide statement when we double click it store the data*  
 CLEAR wa\_vbrk.  
 SKIP.  
 ENDLOOP.  
 CLEAR wa\_vbrk.  
  
AT LINE-SELECTION.  
 READ LINE sy-lilli.  
  
 IF sy-subrc = 0.  
 SELECT \* FROM zmahe\_first02 INTO TABLE it\_vbrp WHERE vbeln EQ wa\_vbrk-vbeln.*"v\_vbeln.*  
 WRITE : / 'Item',  
 20 'Material',  
 40 'Quantity',  
 60 'UoM',  
 90 'Net Value'.  
 LOOP AT it\_vbrp INTO wa\_vbrp.  
 WRITE : / wa\_vbrp-posnr,  
 20 wa\_vbrp-matnr,  
 40 wa\_vbrp-fkimg,  
 60 wa\_vbrp-vrkme,  
 90 wa\_vbrp-netwr.  
  
 ENDLOOP.  
  
 ENDIF.

…………………………………………………………………………………………………………………………………………………………………………………………………

Processing Blocks

Program constructor event

Load-of-program – when program is loaded into the memory.

Anything we want to do at the beginning of the program execution.

Reporting events

: - can be used only in executable type of program and end of the selection

Beginning of the program execution after the program is loaded.

Actually, even if you don't use end of selection, it's okay, like instead of selection. Only if you put the output logic at there, that will be called at the end of the this set of selection is more important when you use that other 2 events region to get tables

Selection screen generated when you are using parameters also and selection options see what that is selection screen. So if you are having that selection scheme for your program, then you can use this selection in the Us..

The selection screen output is called before the selection screen.

PROCEDURES

Procedures have a local data area and can have a parameter interface.

They can be called internally from within the same program or externally from other ABAP program.

Procedures provide reusable software blocks.

Types of procedures.

1. Subroutines
2. Function modules
3. Methods

Subroutines can be implemented between the statements.

Form

…….

Endform.

They are called by the user the perform statement.

…………………………………………………………………………………………..

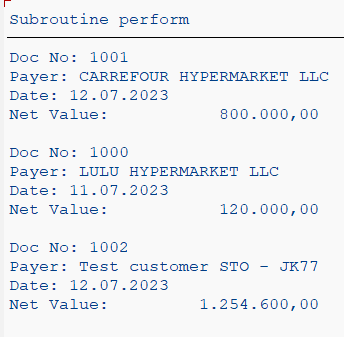
DATA: it\_makt TYPE STANDARD TABLE OF makt.  
  
  
PERFORM get\_data. *"form only execute when we call perform*   
  
form get\_data.  
   
 select \* FROM makt INTO TABLE it\_makt.  
   
 ENDFORM.

………………………………………………………………………………………………..

We can pass parameter in three ways

1. Pass by reference - using and changing
2. Pass by value – using value ex: using value(p\_matnr)
3. Pass value and result

DATA : it\_vbrk TYPE TABLE OF zmahe\_first01,  
 wa\_vbrk LIKE LINE OF it\_vbrk.  
DATA : it\_vbrp TYPE TABLE OF zmahe\_first02,  
 wa\_vbrp LIKE LINE OF it\_vbrp.  
  
DATA : it\_header TYPE zmahe\_headerline.  
  
  
  
  
SELECT-OPTIONS s\_vbeln FOR wa\_vbrk-vbeln.  
START-OF-SELECTION.  
  
*" Data Selection*  
PERFORM get\_header IN PROGRAM Zmahe\_subroutine USING s\_vbeln[]  
 CHANGING it\_header.  
DATA wa\_header LIKE LINE OF it\_header.  
LOOP AT it\_header INTO wa\_header.  
WRITE : / 'Doc No:', wa\_header-vbeln.  
HIDE wa\_header-vbeln.  
 WRITE : / 'Payer:', wa\_header-name1,  
 / 'Date:', wa\_header-fkdat,  
 / 'Net Value:', wa\_header-netwr.  
  
 SKIP.  
 CLEAR wa\_header.  
  
ENDLOOP.  
DATA : it\_item TYPE Zmahe\_itemline.  
  
AT LINE-SELECTION.  
 CLEAR it\_item.  
PERFORM get\_item IN PROGRAM zmahe\_subroutine USING wa\_header-vbeln CHANGING it\_item.  
CLEAR wa\_header.  
DATA wa\_item LIKE LINE OF it\_item.  
  
 WRITE : / 'Item',  
 'Material',  
 'Quantity',  
 'UoM',  
 'Net Value'.  
LOOP AT it\_item INTO wa\_item.  
 READ TABLE it\_header INTO wa\_header WITH KEY vbeln = wa\_item-vbeln.  
 *"READ TABLE it\_t006a INTO wa\_t006a WITH KEY mseht = wa\_item-mseht.*  
 WRITE : / wa\_item-posnr,  
 wa\_item-matnr,  
 wa\_item-fkimg,  
 wa\_item-mseht,  
 wa\_item-netwr.  
 HIDE wa\_item-vbeln.  
 SKIP.  
 CLEAR wa\_item.  
 ENDLOOP.



Subroutine

REPORT ZMAHE\_SUBROUTINE.  
TYPES tr\_vbeln TYPE RANGE OF vbrk-vbeln.  
TYPES ts\_vbeln TYPE RANGE OF vbrp-vbeln.  
  
FORM get\_header USING r\_vbeln TYPE tr\_vbeln CHANGING p\_it\_header TYPE zmahe\_headerline.  
 TYPES : BEGIN OF ty\_kna1,  
 kunnr TYPE kna1-kunnr,  
 name1 TYPE kna1-name1,  
 END OF ty\_kna1.  
DATA : lt\_vbrk TYPE TABLE OF zmahe\_first01,  
 lw\_vbrk LIKE LINE OF lt\_vbrk.  
  
DATA : lt\_kna1 TYPE STANDARD TABLE OF ty\_kna1,  
 lw\_kna1 LIKE LINE OF lt\_kna1.  
  
DATA lw\_header LIKE LINE OF p\_it\_header.  
  
SELECT \* FROM zmahe\_first01 INTO TABLE lt\_vbrk WHERE vbeln IN r\_vbeln.  
 IF sy-subrc IS INITIAL.  
 SELECT kunnr name1 from kna1 INTO TABLE lt\_kna1 FOR ALL ENTRIES IN lt\_vbrk WHERE kunnr EQ lt\_vbrk-kunrg.  
 ENDIF.  
  
 LOOP AT lt\_vbrk INTO lw\_vbrk.  
 READ TABLE lt\_kna1 INTO lw\_kna1 WITH KEY kunnr = lw\_vbrk-kunrg.  
 MOVE-CORRESPONDING lw\_vbrk TO lw\_header.  
 MOVE-CORRESPONDING lw\_kna1 TO lw\_header.  
 APPEND lw\_header TO p\_it\_header.  
 CLEAR lw\_header.  
 ENDLOOP.  
ENDFORM.  
  
FORM get\_item USING p\_vbeln TYPE vbrk-vbeln CHANGING p\_it\_item TYPE zmahe\_itemline.  
 TYPES : BEGIN OF ty\_t006a,  
 msehi TYPE t006a-msehi,  
 mseht TYPE t006a-mseht,  
 END OF ty\_t006a.  
 DATA : lt\_vbrp TYPE TABLE OF zmahe\_first02,  
 lw\_vbrp LIKE LINE OF lt\_vbrp.  
  
 DATA : lt\_t006a TYPE STANDARD TABLE OF ty\_t006a,  
 lw\_t006a LIKE LINE OF lt\_t006a.  
  
 DATA lw\_item LIKE LINE OF p\_it\_item.  
  
 SELECT \* FROM Zmahe\_first02 INTO TABLE lt\_vbrp WHERE vbeln EQ p\_vbeln.  
 IF sy-subrc IS INITIAL.  
 SELECT msehi mseht FROM t006a INTO TABLE lt\_t006a FOR ALL ENTRIES IN lt\_vbrp WHERE msehi EQ lt\_vbrp-vrkme.  
 ENDIF.  
  
 LOOP AT lt\_vbrp INTO lw\_vbrp.  
 READ TABLE lt\_t006a INTO lw\_t006a WITH KEY msehi = lw\_vbrp-vrkme.  
 MOVE-CORRESPONDING lw\_vbrp TO lw\_item.  
 MOVE-CORRESPONDING lw\_t006a TO lw\_item.  
 APPEND lw\_item TO p\_it\_item.  
 CLEAR lw\_item.  
 ENDLOOP.  
ENDFORM.



………………………………………………………………………………………......

Example 2:

TYPES : BEGIN OF ty\_kna1,  
 kunnr TYPE kna1-kunnr,  
 name1 TYPE kna1-name1,  
 END OF ty\_kna1.  
  
TYPES : BEGIN OF ty\_t006a,  
 msehi TYPE t006a-msehi,  
 mseht TYPE t006a-mseht,  
 END OF ty\_t006a.  
*" Data Objects*  
DATA : it\_vbrk TYPE TABLE OF Zmahe\_first01,  
 wa\_vbrk LIKE LINE OF it\_vbrk.  
DATA : it\_vbrp TYPE TABLE OF zmahe\_first02,  
 wa\_vbrp LIKE LINE OF it\_vbrp.  
DATA : it\_header TYPE zmahe\_headerline,  
 it\_item TYPE zmahe\_itemline,  
 wa\_header LIKE LINE OF it\_header,  
 wa\_item LIKE LINE OF it\_item.  
DATA it\_kna1 TYPE STANDARD TABLE OF ty\_kna1.  
DATA it\_t006a TYPE STANDARD TABLE OF ty\_t006a.  
DATA : wa\_kna1 LIKE LINE OF it\_kna1,  
 wa\_t006a LIKE LINE OF it\_t006a.  
  
*" Provide selection screen for user Input*  
  
START-OF-SELECTION.  
 SELECT-OPTIONS s\_vbeln FOR wa\_vbrk-vbeln.  
 *" Data Selection*  
 PERFORM get\_header IN PROGRAM zmahe\_subroutine01 USING s\_vbeln[]  
 CHANGING it\_header.  
  
 PERFORM get\_item IN PROGRAM zmahe\_subroutine01 USING s\_vbeln[]  
 CHANGING it\_item.  
  
  
 WRITE : / 'Doc No:',15 'Payer:',45 'Date:',65 'Net Value:'.  
  
  
*\* SORT it\_item BY vbeln.*  
 loop AT it\_header INTO wa\_header.  
  
  
 SKIP.  
*\* READ TABLE it\_header INTO wa\_header WITH KEY vbeln = wa\_item-vbeln.*  
 WRITE : / wa\_header-vbeln,  
 wa\_header-name1,  
 wa\_header-fkdat,  
 wa\_header-netwr.  
 HIDE wa\_header-vbeln.  
 SKIP.  
 CLEAR wa\_header.  
ENDLOOP.  
CLEAR wa\_header.  
  
AT LINE-SELECTION.  
 WRITE : / 'Item',  
 10 'Material',  
 40 'Quantity',  
 50'UoM',  
 70 'Net Value'.  
  
 LOOP AT it\_item INTO wa\_item WHERE vbeln EQ wa\_header-vbeln.  
  
 WRITE : / wa\_item-posnr,  
 10 wa\_item-matnr LEFT-JUSTIFIED,  
 40 wa\_item-fkimg LEFT-JUSTIFIED,  
 50 wa\_item-mseht LEFT-JUSTIFIED,  
 70 wa\_item-netwr LEFT-JUSTIFIED.  
 ENDLOOP.

Subroutine

TYPES tr\_vbeln TYPE RANGE OF vbrk-vbeln.  
  
FORM get\_header USING r\_vbeln TYPE tr\_vbeln CHANGING p\_it\_header TYPE zmahe\_headerline.  
 TYPES : BEGIN OF ty\_kna1,  
 kunnr TYPE kna1-kunnr,  
 name1 TYPE kna1-name1,  
 END OF ty\_kna1.  
 DATA : lt\_vbrk TYPE TABLE OF Zmahe\_first01,  
 lw\_vbrk LIKE LINE OF lt\_vbrk.  
  
 DATA : lt\_kna1 TYPE STANDARD TABLE OF ty\_kna1,  
 lw\_kna1 LIKE LINE OF lt\_kna1.  
  
 DATA lw\_header LIKE LINE OF p\_it\_header.  
  
 SELECT \* FROM Zmahe\_first01 INTO TABLE lt\_vbrk WHERE vbeln IN r\_vbeln.  
 IF sy-subrc IS INITIAL.  
 SELECT kunnr name1 FROM kna1 INTO TABLE lt\_kna1 FOR ALL ENTRIES IN lt\_vbrk WHERE kunnr EQ lt\_vbrk-kunrg.  
 ENDIF.  
  
 LOOP AT lt\_vbrk INTO lw\_vbrk.  
 READ TABLE lt\_kna1 INTO lw\_kna1 WITH KEY kunnr = lw\_vbrk-kunrg.  
 MOVE-CORRESPONDING lw\_vbrk TO lw\_header.  
 MOVE-CORRESPONDING lw\_kna1 TO lw\_header.  
 APPEND lw\_header TO p\_it\_header.  
 CLEAR lw\_header.  
 ENDLOOP.  
ENDFORM.  
  
FORM get\_item USING r\_vbeln TYPE tr\_vbeln CHANGING p\_it\_item TYPE Zmahe\_itemline.  
 TYPES : BEGIN OF ty\_t006a,  
 msehi TYPE t006a-msehi,  
 mseht TYPE t006a-mseht,  
 END OF ty\_t006a.  
 DATA : lt\_vbrp TYPE TABLE OF Zmahe\_first02,  
 lw\_vbrp LIKE LINE OF lt\_vbrp.  
 DATA : lt\_t006a TYPE STANDARD TABLE OF ty\_t006a,  
 lw\_t006a LIKE LINE OF lt\_t006a.  
  
 DATA lw\_item LIKE LINE OF p\_it\_item.  
  
 SELECT \* FROM zmahe\_first02 INTO TABLE lt\_vbrp WHERE vbeln IN r\_vbeln.  
 IF sy-subrc IS INITIAL.  
 SELECT msehi mseht FROM t006a INTO TABLE lt\_t006a FOR ALL ENTRIES IN lt\_vbrp WHERE msehi EQ lt\_vbrp-vrkme.  
 ENDIF.  
 LOOP AT lt\_vbrp INTO lw\_vbrp.  
 READ TABLE lt\_t006a INTO lw\_t006a WITH KEY msehi = lw\_vbrp-vrkme.  
 MOVE-CORRESPONDING lw\_vbrp TO lw\_item.  
 MOVE-CORRESPONDING lw\_t006a TO lw\_item.  
 APPEND lw\_item TO p\_it\_item.  
 CLEAR lw\_item.  
 ENDLOOP.  
  
  
ENDFORM.  
  
  
FORM get\_billing\_data USING r\_vbeln TYPE tr\_vbeln CHANGING p\_it\_header TYPE zdemo\_t\_header p\_it\_item TYPE zmahe\_itemline.  
 TYPES : BEGIN OF ty\_kna1,  
 kunnr TYPE kna1-kunnr,  
 name1 TYPE kna1-name1,  
 END OF ty\_kna1.  
 DATA : lt\_vbrk TYPE TABLE OF Zmahe\_first01,  
 lw\_vbrk LIKE LINE OF lt\_vbrk.  
  
 DATA : lt\_kna1 TYPE STANDARD TABLE OF ty\_kna1,  
 lw\_kna1 LIKE LINE OF lt\_kna1.  
  
 DATA lw\_header LIKE LINE OF p\_it\_header.  
  
 SELECT \* FROM Zmahe\_first01 INTO TABLE lt\_vbrk WHERE vbeln IN r\_vbeln.  
 IF sy-subrc IS INITIAL.  
 SELECT kunnr name1 FROM kna1 INTO TABLE lt\_kna1 FOR ALL ENTRIES IN lt\_vbrk WHERE kunnr EQ lt\_vbrk-kunrg.  
 ENDIF.  
  
 LOOP AT lt\_vbrk INTO lw\_vbrk.  
 READ TABLE lt\_kna1 INTO lw\_kna1 WITH KEY kunnr = lw\_vbrk-kunrg.  
 MOVE-CORRESPONDING lw\_vbrk TO lw\_header.  
 MOVE-CORRESPONDING lw\_kna1 TO lw\_header.  
 APPEND lw\_header TO p\_it\_header.  
 CLEAR lw\_header.  
 ENDLOOP.  
  
 TYPES : BEGIN OF ty\_t006a,  
 msehi TYPE t006a-msehi,  
 mseht TYPE t006a-mseht,  
 END OF ty\_t006a.  
 DATA : lt\_vbrp TYPE TABLE OF Zmahe\_first02,  
 lw\_vbrp LIKE LINE OF lt\_vbrp.  
 DATA : lt\_t006a TYPE STANDARD TABLE OF ty\_t006a,  
 lw\_t006a LIKE LINE OF lt\_t006a.  
  
 DATA lw\_item LIKE LINE OF p\_it\_item.  
  
 SELECT \* FROM zmahe\_first02 INTO TABLE lt\_vbrp WHERE vbeln IN r\_vbeln.  
 IF sy-subrc IS INITIAL.  
 SELECT msehi mseht FROM t006a INTO TABLE lt\_t006a FOR ALL ENTRIES IN lt\_vbrp WHERE msehi EQ lt\_vbrp-vrkme.  
 ENDIF.  
 LOOP AT lt\_vbrp INTO lw\_vbrp.  
 READ TABLE lt\_t006a INTO lw\_t006a WITH KEY msehi = lw\_vbrp-vrkme.  
 MOVE-CORRESPONDING lw\_vbrp TO lw\_item.  
 MOVE-CORRESPONDING lw\_t006a TO lw\_item.  
 APPEND lw\_item TO p\_it\_item.  
 CLEAR lw\_item.  
 ENDLOOP.  
  
ENDFORM.

When we call the subroutine, we can pass some parameters to it. It also supports some local data so we can do some data.

**FUNCTION MODULE**

Functions also same maintain between function and function by using call function.

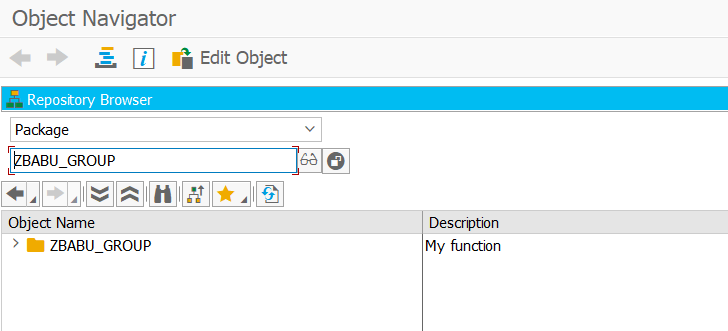
We can pass parameters to a function module.

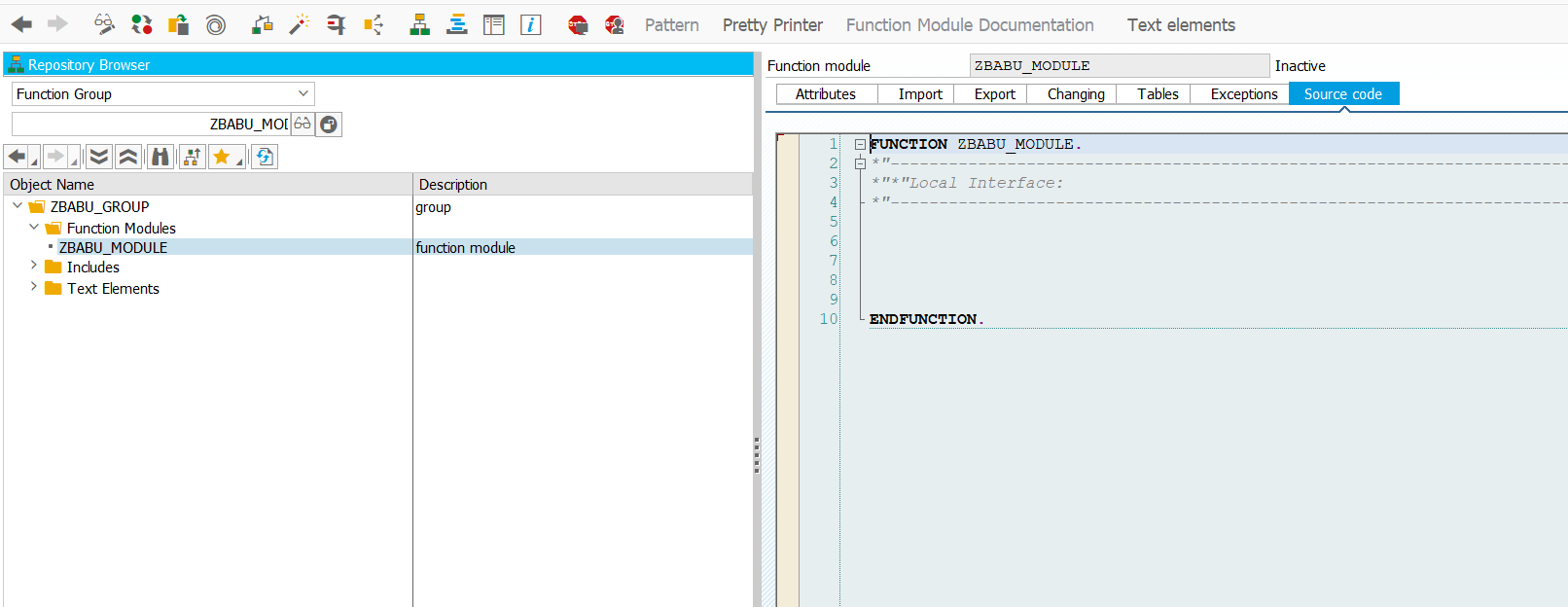
We can create function subroutine without passing any parameter into it.

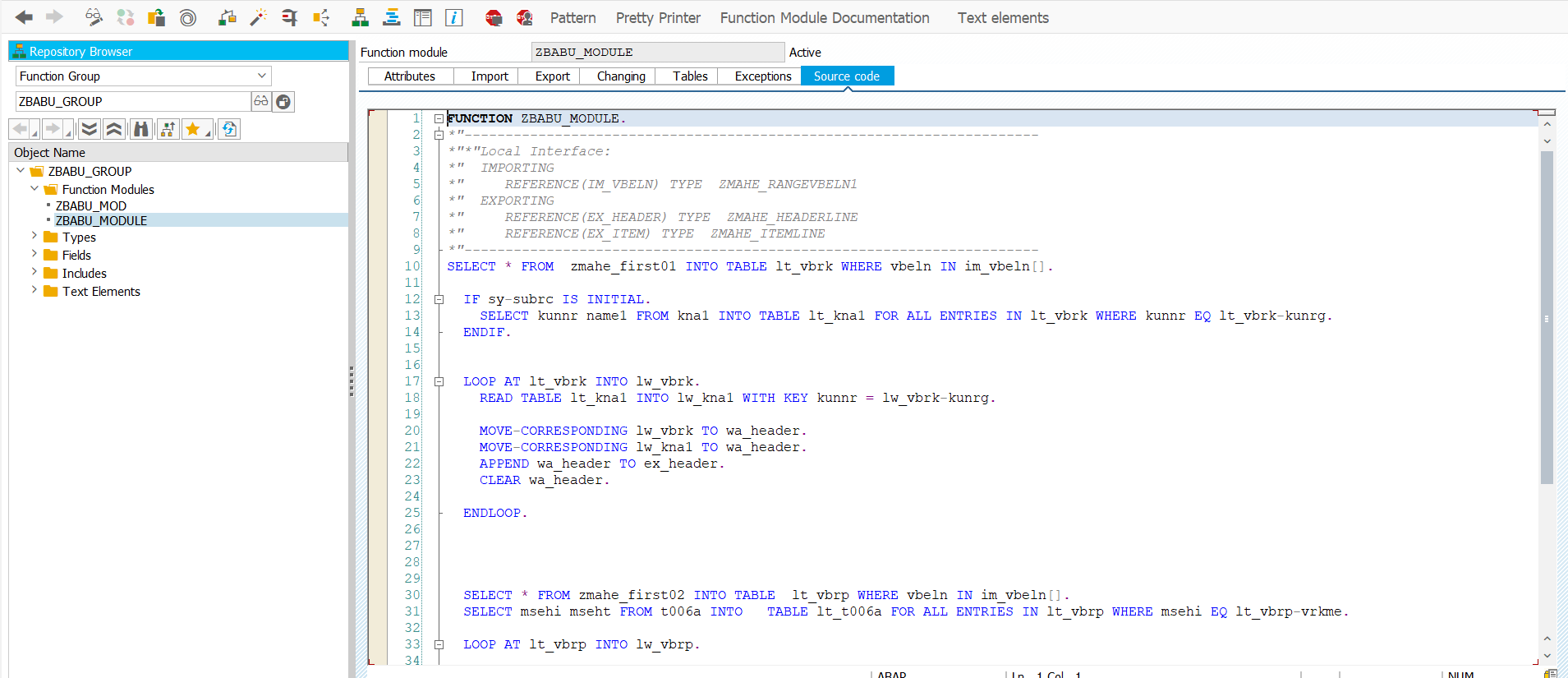
Function module can’t create in a executable program.

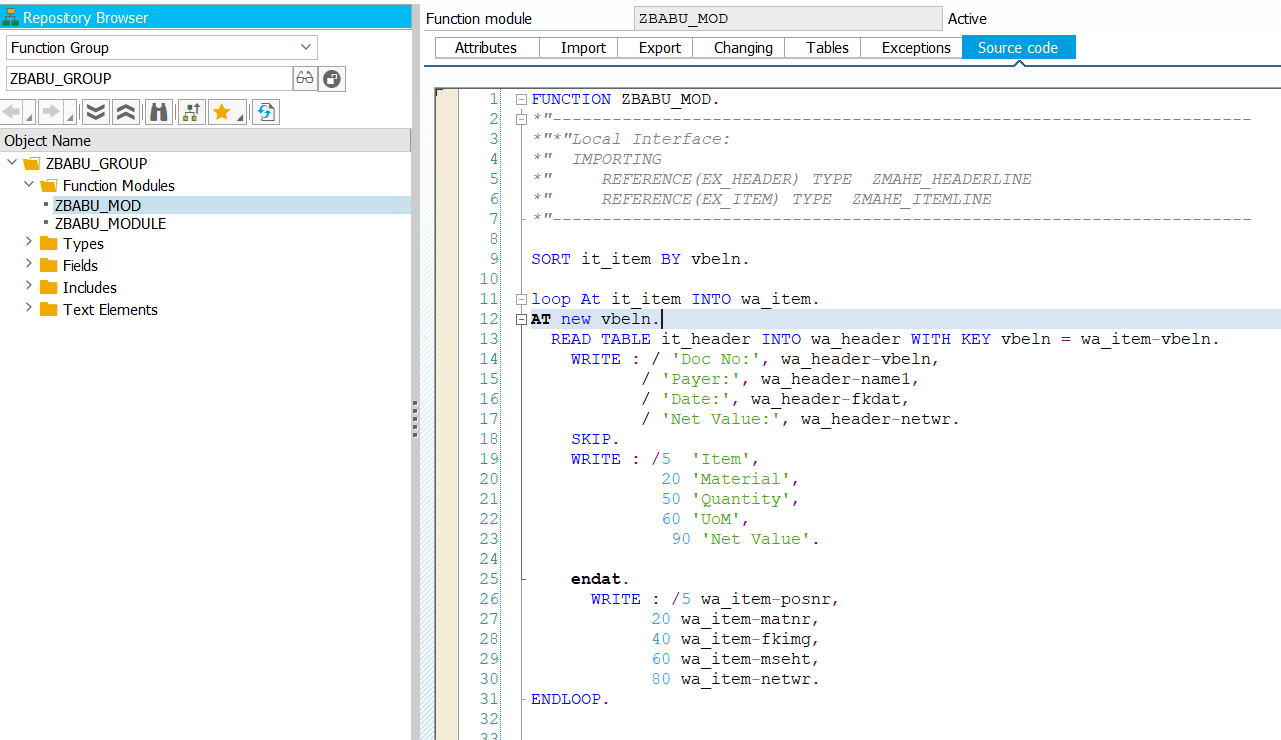
We have to create another type of program is called function pool.

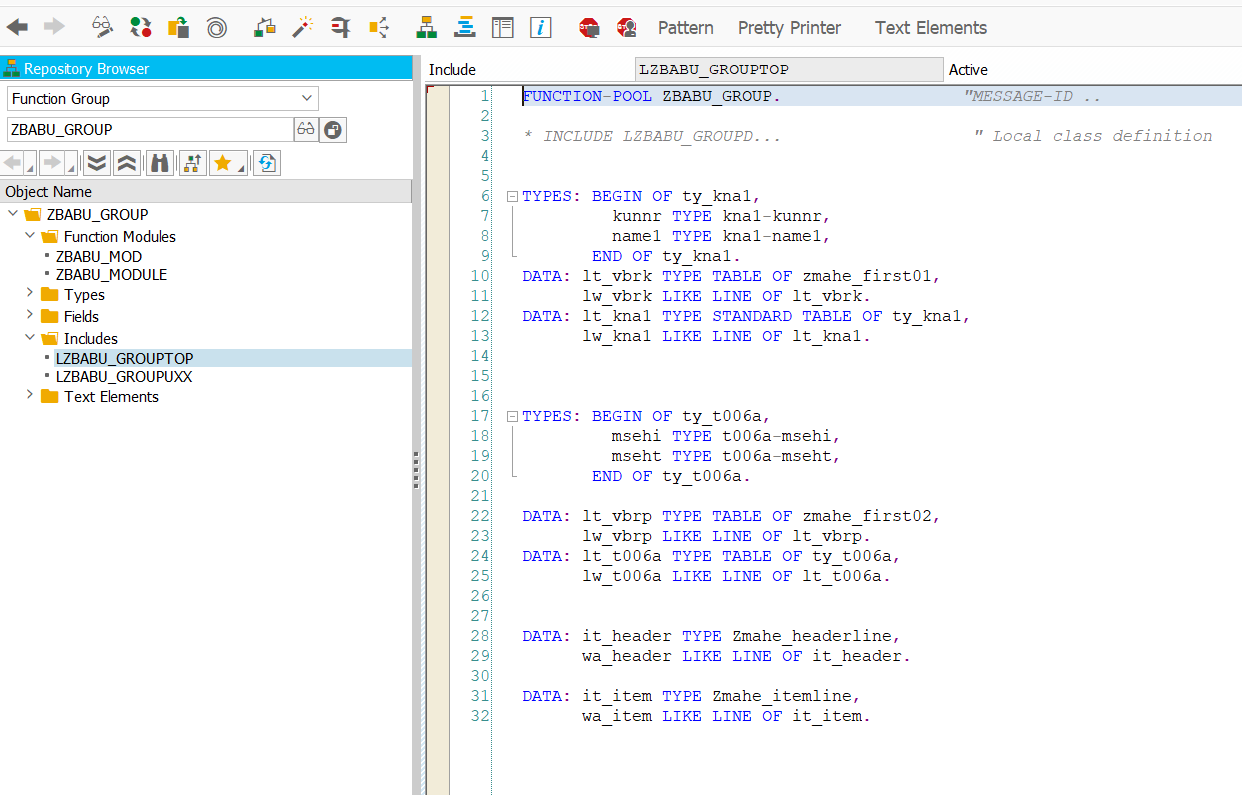
To create Function group and Function module we use se80.

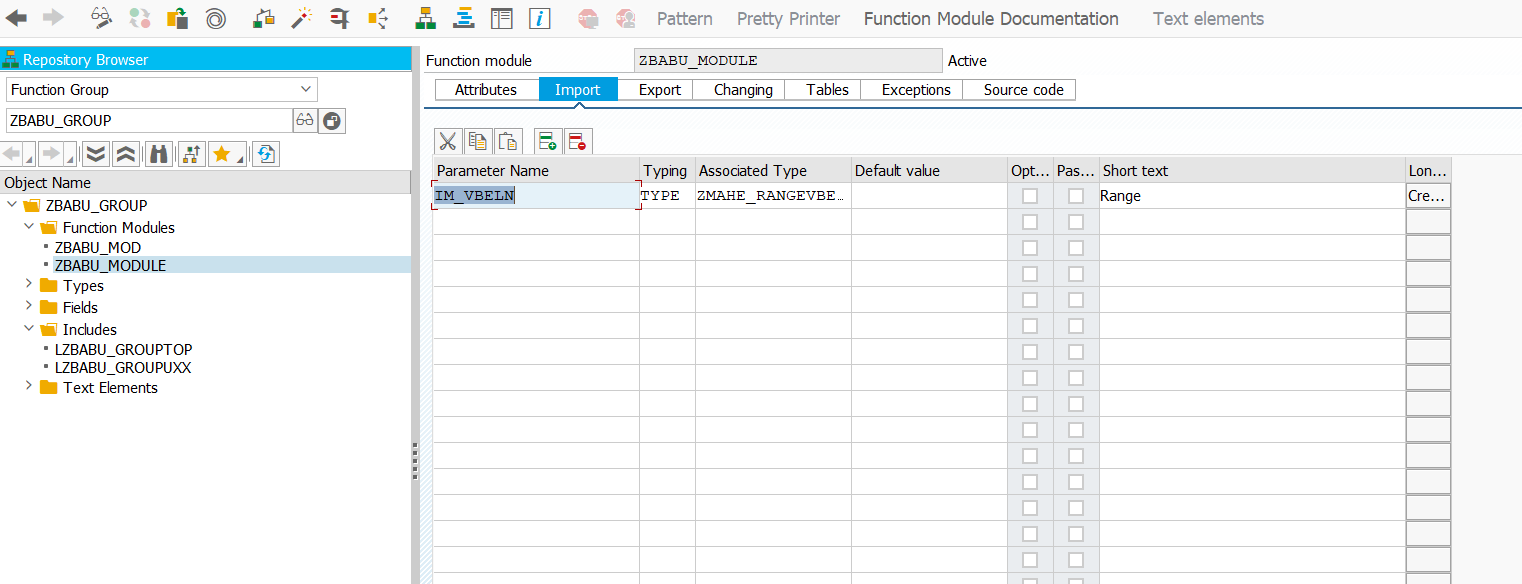


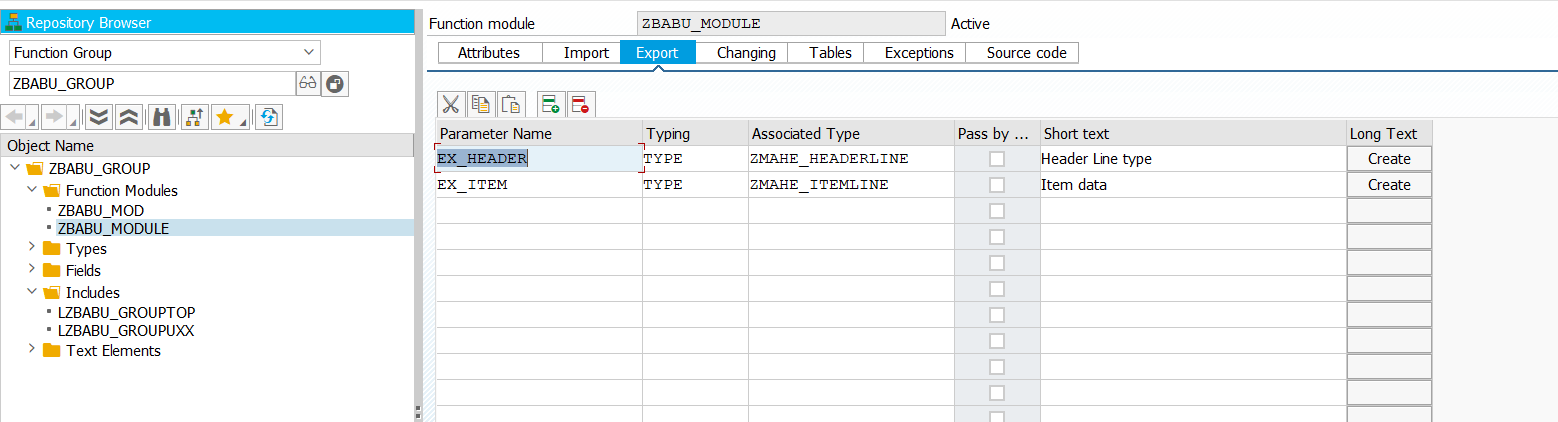




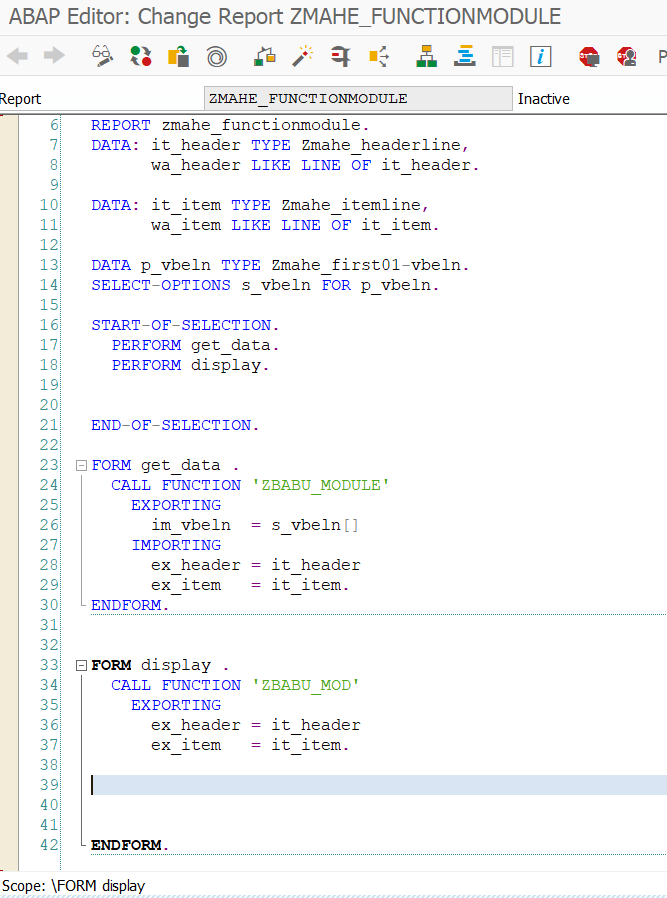












WE create a program to upload a data.

In TXT

123 Mahesh 9066774471

456 Rohit 9639639396

TYPES: BEGIN OF ty\_file,  
 f1 TYPE char10,  
 f2 TYPE char10,  
 f3 TYPE char10,  
 END OF ty\_file.  
  
DATA: it\_file TYPE STANDARD TABLE OF ty\_file,  
 wa\_file LIKE LINE OF it\_file.  
data v\_doc TYPE string.  
PARAMETERS p\_file TYPE IBIPPARMS-PATH.  
  
AT SELECTION-SCREEN ON VALUE-REQUEST FOR p\_file.  
 CALL FUNCTION 'F4\_FILENAME'  
 EXPORTING  
 PROGRAM\_NAME = SYST-CPROG  
 DYNPRO\_NUMBER = SYST-DYNNR  
 FIELD\_NAME = 'P\_FILE'  
 IMPORTING  
 FILE\_NAME = p\_file.  
START-OF-SELECTION.  
v\_doc = p\_file.  
CALL FUNCTION 'GUI\_UPLOAD'  
 EXPORTING  
 filename = v\_doc “Here we are taking vdoc because this filename take as string if we give p\_file it throws error  
 filetype = 'ASC'  
 has\_field\_separator = 'x'  
*\* HEADER\_LENGTH = 0*  
*\* READ\_BY\_LINE = 'X'*  
*\* DAT\_MODE = ' '*  
*\* CODEPAGE = ' '*  
*\* IGNORE\_CERR = ABAP\_TRUE*  
*\* REPLACEMENT = '#'*  
*\* CHECK\_BOM = ' '*  
*\* VIRUS\_SCAN\_PROFILE =*  
*\* NO\_AUTH\_CHECK = ' '*  
*\* IMPORTING*  
*\* FILELENGTH =*  
*\* HEADER =*  
 TABLES  
 data\_tab = it\_file  
*\* CHANGING*  
*\* ISSCANPERFORMED = ' '*  
 EXCEPTIONS  
 file\_open\_error = 1  
 file\_read\_error = 2  
 no\_batch = 3  
 gui\_refuse\_filetransfer = 4  
 invalid\_type = 5  
 no\_authority = 6  
 unknown\_error = 7  
 bad\_data\_format = 8  
 header\_not\_allowed = 9  
 separator\_not\_allowed = 10  
 header\_too\_long = 11  
 unknown\_dp\_error = 12  
 access\_denied = 13  
 dp\_out\_of\_memory = 14  
 disk\_full = 15  
 dp\_timeout = 16  
 OTHERS = 17.  
IF sy-subrc <> 0.  
*\* Implement suitable error handling here*  
  
 LOOP AT it\_file INTO wa\_file.  
 WRITE:/ wa\_file-f1,wa\_file-f2,wa\_file-f3.  
 ENDLOOP.  
ENDIF.

………………………………………………………………………………………………………………………………………………………………………………………………………

OOPS CONCEPT

Objects can interact with one another by sending messages. Objects contain data and code to manipulate the data. An object can also be used as a user-defined data type with the help of a class. Objects are also called variables of the type class. After defining a class, you can create any number of objects belonging to that class. Each object is associated with the data of the type class with which it has been created.

Creating an Object

The object creation usually includes the following steps −

* Creating a reference variable with reference to the class. The syntax for which is −

**DATA: <object\_name> TYPE REF TO <class\_name>.**

* Creating an object from the reference variable. The syntax for which is −

CREATE Object: <object\_name>.

CLASS Class1 DEFINITION.  
 PUBLIC SECTION.  
 DATA: text1(45) VALUE 'ABAP Objects.'. *"Here we specify the data type as text.*  
 METHODS: Display1. *"Here we declaring method*  
ENDCLASS.  
  
CLASS Class1 IMPLEMENTATION.  
 METHOD Display1.  
 WRITE:/ 'This is the Display method.'. *"Here we implementing data to the methods*  
 ENDMETHOD.  
ENDCLASS.  
  
START-OF-SELECTION.  
 DATA: Class1 TYPE REF TO Class1. *"here we object refer to the class name*  
 CREATE OBJECT: Class1. *"Here we creating object for reference class*  
 WRITE:/ Class1->text1.  
 CALL METHOD: Class1->Display1.

………………………………………………………………………………………………………………………………………………………………………………………………………

A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package. The data and functions within a class are called **members of the class**.

## Class Definition and Implementation

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, what an object of the class will consist of, and what operations can be performed on such an object. That is, it defines the abstract characteristics of an object, such as attributes, fields, and properties.

The following syntax shows how to define a class −

CLASS <class\_name> DEFINITION.

..........

..........

ENDCLASS.

A class definition starts with the keyword CLASS followed by the class name, DEFINITION and the class body. The definition of a class can contain various components of the class such as attributes, methods, and events. When we declare a method in the class declaration, the method implementation must be included in the class implementation. The following syntax shows how to implement a class −

CLASS <class\_name> IMPLEMENTATION.

...........

..........

ENDCLASS.

**Note** − Implementation of a class contains the implementation of all its methods. In ABAP Objects, the structure of a class contains components such as attributes, methods, events, types, and constants.

## Attributes

Attributes are data fields of a class that can have any data type such as C, I, F, and N. They are declared in the class declaration. These attributes can be divided into 2 categories: instance and static attributes. An **instance attribute** defines the instance specific state of an object. The states are different for different objects. An instance attribute is declared by using the DATA statement.

**Static attributes** define a common state of a class that is shared by all the instances of the class. That is, if you change a static attribute in one object of a class, the change is visible to all other objects of the class as well. A static attribute is declared by using the CLASS-DATA statement.

## Methods

A method is a function or procedure that represents the behavior of an object in the class. The methods of the class can access any attribute of the class. The definition of a method can also contain parameters, so that you can supply the values to these parameters when methods are called. The definition of a method is declared in the class declaration and implemented in the implementation part of a class. The METHOD and ENDMETHOD statements are used to define the implementation part of a method. The following syntax shows how to implement a method −

METHOD <m\_name>.

..........

..........

ENDMETHOD.

In this syntax, <m\_name> represents the name of a method. **Note** − You can call a method by using the CALL METHOD statement.

## Accessing Attributes and Methods

Class components can be defined in public, private, or protected visibility sections that control how these components could be accessed. The private visibility section is used to deny access to components from outside of the class. Such components can only be accessed from inside the class such as a method.

Components defined in the public visibility section can be accessed from any context. By default all the members of a class would be private. Practically, we define data in private section and related methods in public section so that they can be called from outside of the class as shown in the following program.

* The attributes and methods declared in Public section in a class can be accessed by that class and any other class, sub-class of the program.
* When the attributes and methods are declared in Protected section in a class, those can be accessed by that class and sub classes (derived classes) only.
* When the attributes and methods are declared in Private section in a class, those can be accessed by only that class and not by any other class.

### **Example**

Report ZAccess1.

CLASS class1 Definition.

PUBLIC Section.

Data: text1 Type char25 Value 'Public Data'.

Methods meth1.

PROTECTED Section.

Data: text2 Type char25 Value 'Protected Data'.

PRIVATE Section.

Data: text3 Type char25 Value 'Private Data'.

ENDCLASS.

CLASS class1 Implementation.

Method meth1.

Write: / 'Public Method:',

/ text1,

/ text2,

/ text3.

Skip.

EndMethod.

ENDCLASS.

Start-Of-Selection.

Data: Objectx Type Ref To class1.

Create Object: Objectx.

CALL Method: Objectx→meth1.

Write: / Objectx→text1.

The above code produces the following output −

Public Method:

Public Data

Protected Data

Private Data

Public Data

## Static Attributes

A Static attribute is declared with the statement CLASS-DATA. All the objects or instances can use the static attribute of the class. Static attributes are accessed directly with the help of class name like class\_name⇒name\_1 = 'Some Text'.

### **Example**

Following is a program where we want to print a text with line number 4 to 8 times. We define a class class1 and in the public section we declare CLASS-DATA (static attribute) and a method. After implementing the class and method, we directly access the static attribute in Start-Of-Selection event. Then we just create the instance of the class and call the method.

Report ZStatic1.

CLASS class1 Definition.

PUBLIC Section.

CLASS-DATA: name1 Type char45,

data1 Type I.

Methods: meth1.

ENDCLASS.

CLASS class1 Implementation.

Method meth1.

Do 4 Times.

data1 = 1 + data1.

Write: / data1, name1.

EndDo.

Skip.

EndMethod.

ENDCLASS.

Start-Of-Selection.

class1⇒name1 = 'ABAP Object Oriented Programming'.

class1⇒data1 = 0.

Data: Object1 Type Ref To class1,

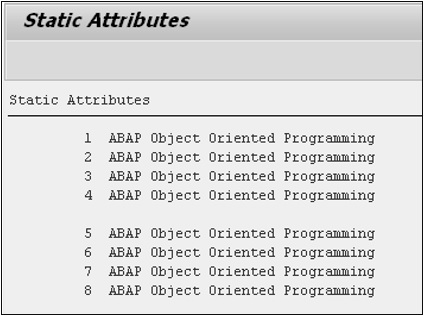
Object2 Type Ref To class1.

Create Object: Object1, Object2.

CALL Method: Object1→meth1,

Object2→meth1.

The above code produces the following output −



## Constructors

Constructors are special methods that are called automatically, either while creating an object or accessing the components of a class. Constructor gets triggered whenever an object is created, but we need to call a method to trigger the general method. In the following example, we have declared two public methods method1 and constructor. Both these methods have different operations. While creating an object of the class, the constructor method triggers its operation.

### **Example**

Report ZConstructor1.

CLASS class1 Definition.

PUBLIC Section.

Methods: method1, constructor.

ENDCLASS.

CLASS class1 Implementation.

Method method1.

Write: / 'This is Method1'.

EndMethod.

Method constructor.

Write: / 'Constructor Triggered'.

EndMethod.

ENDCLASS.

Start-Of-Selection.

Data Object1 Type Ref To class1.

Create Object Object1.

The above code produces the following output −

Constructor Triggered

## ME Operator in Methods

When you declare a variable of any type in public section of a class, you can use it in any other implementation. A variable can be declared with an initial value in public section. We may declare the variable again inside a method with a different value. When we write the variable inside the method, the system will print the changed value. To reflect the previous value of the variable, we have to use ‘ME’ operator.

In this program, we have declared a public variable text1 and initiated with a value. We have declared the same variable again, but instantiated with different value. Inside the method, we are writing that variable with ‘ME’ operator to get the previously initiated value. We get the changed value by declaring directly.

### **Example**

Report ZMEOperator1.

CLASS class1 Definition.

PUBLIC Section.

Data text1 Type char25 Value 'This is CLASS Attribute'.

Methods method1.

ENDCLASS.

CLASS class1 Implementation.

Method method1.

Data text1 Type char25 Value 'This is METHOD Attribute'.

Write: / ME→text1,

/ text1.

ENDMethod.

ENDCLASS.

Start-Of-Selection.

Data objectx Type Ref To class1.

Create Object objectx.

CALL Method objectx→method1.

The above code produces the following output −

This is CLASS Attribute

This is METHOD Attribute

……………………………………………………………………………………………………………………………………………………………………

Class – is the template or blueprint.

Object- Runtime instance of class.

Class Components.

Attributes.

Methods.

Events.

Example 1:

Class lcl DEFINITION.  
 PUBLIC SECTION.  
 data count type i.  
 METHODS next\_number EXPORTING number TYPE i.  
 PROTECTED SECTION.  
 PRIVATE SECTION.  
  
 ENDCLASS.  
  
 class lcl IMPLEMENTATION.  
 method next\_number.  
 number = count + 1.  
 count = number.  
 ENDMETHOD.  
 ENDCLASS.  
  
 START-OF-SELECTION.  
 data:mahe TYPE REF TO lcl.  
 CREATE OBJECT:mahe.  
  
 data v\_doc TYPE i.  
 mahe->next\_number(  
 IMPORTING  
 number = v\_doc  
 ).

This ABAP code defines a class called **lcl** with three sections: **PUBLIC**, **PROTECTED**, and **PRIVATE**.

The **PUBLIC** section contains a data member called **count** of type **i** (integer) and a method **next\_number**.

**count**: It is a variable that will be used to store and keep track of the next number to be generated by the method.

**next\_number**: It is a method without any explicit parameters to be passed from the caller. However, it has an exporting parameter called **number** of type **i**. This means that when the method is called, it will provide the value of the next number through the **number** parameter.

The **PROTECTED** and **PRIVATE** sections are currently empty, as no specific data members or methods have been declared within them.

Next, in the **class lcl IMPLEMENTATION** section, the method **next\_number** is defined.

The **next\_number** method is implemented to generate the next number in the sequence. It increments the value of **count** by 1 and assigns the incremented value to the exporting parameter **number**.

In the **START-OF-SELECTION** section, the main program logic begins:

It declares a variable **mahe** of type reference to **lcl**, which means **mahe** will point to an object of the class **lcl**.

It then creates an object of the **lcl** class and assigns the reference to the variable **mahe**.

Next, a variable **v\_doc** of type **i** (integer) is declared to hold the result of calling the **next\_number** method of the **mahe** object.

Finally, the **next\_number** method of the **mahe** object is called, passing the variable **v\_doc** as the exporting parameter **number**. This call will generate the next number in the sequence and store it in the **v\_doc** variable.

To summarize, this code sets up a class with a method to generate the next number in a sequence. It uses an internal variable **count** to keep track of the current number, and each time the **next\_number** method is called, it increments **count** and returns the incremented value as the result. The main program creates an object of this class and retrieves the next number using the **next\_number** method.

Every object will have its own in of that memory. Or that you book on method, whatever only one instant that exist in the entire program memory.

If we define attribute as static can be accessed from any object of that class in the program. If you define an as an instance, attribute, then it resides within that object.

**Introduction:**

**Procedure oriented / structured programming / modular programming languages** ---> C, PASCAL, COBOL --> Importance is given to Functions

**Object oriented** --> ABAP, C++, JAVA, .Net

---> Importance is given to securing the data

**Features of OOPS Oriented Programming Languages**

1. **Encapsulation** --> binding data and member functions into single unit (class)

2. **Data abstraction / Data hiding** --> hiding the implementation details

3. **Inheritance** ---> reusability (inheriting the components of parent class into child class)

4. **Polymorphism-**-> many forms behavior

**Procedure followed for Basic ABAP Report development:**

1. Declare variables, work areas, internal tables...

2. Generate selection screen for reading user input (parameters/select-options)

3. Validate the user input

4. Execute select queries to read data from db into work area / internal table

5. Process the work area/ internal table to show the content (data) to the user

**Types Declaration Vs class declaration**

types : begin of <type name>, (ty\_emp) similar to -------> class declaration

field 1, empno components

field 2, ename

----

end of <type name>. endclass.

ty\_emp-empno = 1. (invalid)

data wa\_emp1 type ty\_emp. similar to --> object1 based on class

wa\_emp1-empno = 1.

wa\_emp1-ename = 'Raju'.

data wa\_emp2 type ty\_emp. --> object2 based on class

wa\_emp2-empno = 4.

wa\_emp2-ename = 'Ramesh'.

**Class** ---> is a User defined data type which is a collection of components

C++ class ---> data members + member functions

Java class --> instance variables + methods

ABAP class ---> attributes + methods + events + interfaces + aliases + macros

ABAP ---> 2 types of classes

1. Local class--> local to an object (program) ---> ABAP editor (se38)

2. Global class --> global to all objects ----> class builder tool (se24)

---> stored inside class pool

**Procedure for creating local class:**

**1. Definition of class**

Syntax:

class <class name> definition [public] [protected] [private]

[deferred] [final] [load] [abstract].

declaration of components.

endclass.

**Note:** If the class definition contains method declarations, then we need to implement the class

**2. Implementation of class**

Syntax:

class <class name> implementation.

implementation of methods.

endclass.

**Note:** Class definition and Class implementation doesn't allocate any memory, it only provides template of the class

**Instantiate the class** ---> creating the object for the class--> memory will be allocated

Object creation --> 2 steps

1. Declare reference (alias) for the class

Syntax:

data <ref.name> type ref to <class name>.

**Note:** reference doesn't allocate any memory, it only provides alias.

2. Create object based on reference

Syntax:

create object <ref.name>. --> Memory will be allocated based on components of class

**Attribute:** is like a variable which can be used to store the data.

**Instance Attributes:**

In local classes they are declared by using the keyword “Data”.

They are specific to object i.e., for every object, separate memory will be allocated for each instance attribute.

They can be accessed only by using the Object.

**Static Attributes:**

In local classes they are declared by using the keyword “class-data”.

They are also called as “Class Variables / Class attributes”.

They are not specific to any object.

The memory for static attributes will be allocated whenever the class is loaded in to the memory i.e. memory will be allocated only once which will be shared by all the objects of the class.

They can be accessed either by using the object or by using the class name.

**Constant Attributes:**

In Local classes, they are declared by using the keyword ‘constants’. They must be initialized at the time of declaration itself. The value of the constant attribute remains same throughout the program.

They are not specific to any object.

The memory for them will be allocated whenever the class is loaded in to the memory i.e. memory will be allocated only once which will be shared by all the objects of the class.

They can be accessed either by using the object or by using the class name.

**Note:** We go for instance attributes whenever we need to maintain unique (different) values for each object. We go for static attributes, whenever we need to maintain a common (same) value for all the objects of the class. We go for constant attributes, whenever we need to maintain a fixed value for all the objects of the class.

**Methods:**

* A method is a set of statements which is implemented only once and can be called any no. of times.
* It is like subroutine / function module.

**Subroutines: -** 2 sections ---> definition ---> form...endform

Calling ---> perform <subroutine>.

Subroutine Parameters 🡪 keywords used 🡪 using, changing, tables

**F.M's:** 2 sections ---> definition --> function..endfunction (source code tab)

calling ---> call function <f.m>

Function Module Parameters 🡪 keywords used 🡪 importing, exporting, changing, tables

**Local class methods:** 3 steps

Declaration (Method prototype declaration) --> Inside Class definition

Implementation --> Inside Class Implementation

Calling --> outside class / inside other method implementation

**Method Parameters:**

**Types of parameters:** importing, exporting, changing and returning.

By Default, Importing parameters are obligatory.

We can use the keyword ‘optional’ as part of local class method parameters to declare it as optional parameter and in case of global class methods, select the checkbox ‘optional’

Exporting parameters are always optional.

**Procedure for using Local class methods:**

1. Declaration

**syntax:**

methods/class-methods <method name> [parameters list].

2. Implementation

**syntax:**

method <method name>.

statements.

endmethod.

3. Calling

**syntax 1:**

call method <method name> [parameters list]

**syntax 2:**

<method name>( [parameters list] ).

**Method Returning Values:**

* In case of ABAP a method can return any no. of values. The no. of return values depends on no of Exporting/changingParameters.

**Returning Parameters:**

* In general, a method can return any no. of parameters.
* To restrict a method to return exactly one value, we must use returning parameter.
* If a method contains returning parameter, it cannot contain exporting /changing parameters at the same time or vice versa (prior EHP7)
* A method can contain only one returning parameter
* Returning parameter is always passed by value.
* Returning parameters are always optional.

**Instance methods:**

In local classes they are declared by using the keyword “Methods”.

They can be accessed only by using the object.

They can access any kind of components of the class (instance / static / constant / types).

**Static methods:**

In local classes they are declared by using the keyword “class-Methods”.

They can be accessed either by using the object or class name.

They can access only static components, constant attributes and types of attributes. i.e. they cannot access instance components.

**Note:** In Local classes, the sequence of visibility sections (access specifiers) for the class components should be in the order of public, protected, private sections.

**Note:** It is recommended to define and implement the local class at the beginning of executable program and explicitly handle the event ‘start-of-selection’ after the class implementation to indicate the starting point of program execution.

**Me keyword:**

“Me” keyword refers to current object in execution i.e., as part of every instance method execution (runtime), an implicitly created object (created by SAP) will be available and it refers to the object which is currently executing the method.

**Note:** If a class contains both attributes and methods, it is recommended to declare attributes under protected/private sections and methods under public sections.

Since we cannot access protected/private components outside the class, we need to access them through one of the public methods and call the public method outside the class.

**Friend keyword:**

In General outside the class, A Object can access only public components of the class directly. i.e protected and private components of the class cannot be accessed outside the class using an object of a class.

In order to enable the objects of the class to access all components of the class outside the class irrespective of the visibility, then we can go for Friend classes.

In local classes “Friend” is a keyword used as part of class definition to consider another class as friend.

**Deferred Keyword**:

As part of declaring local friend classes, we need to forward declare the friend classes by using **“Deferred” keyword.** This deferred keyword indicates that the class definition is provided somewhere else in the program and not at the beginning of the program.

**Note:** In Global classes, we declare friend classes as part of ‘FRIENDS’ tab.

**Load keyword:**

It is used to load the global classes or global interfaces explicitly in the executable program. This is mandatory before the release 6.20 for accessing static components of the global class or global interface before instantiating them.

From 6.20, it is not required, but in case, if we get any compilation error while accessing the static components / any other components of global class or global interface, then we can load the global class or global interface explicitly from the class library by using load keyword.

**Constructor:**

* It is special method used for initializing the attributes of the class i.e., whenever an object is created, the attributes should be initialized with some initial values.
* It is special because it is executed/called automatically whenever an object is created (instance const) or whenever a class is loaded in the memory (static const).
* They are always declared in public section.
* They never return any value.
* There are two types of constructors

1. Instance constructor.
2. Static constructor.

**Instance constructor**

* They are declared by using the keyword “Constructor”.
* They can contain only importing parameters and exceptions.
* It is specific to object i.e. whenever a new object is created SAP executes “Instance constructor”.
* Instance constructor is executed only once in the life time of every object.

**Static constructor**

* It is declared by using keyword “class constructor”.
* It cannot contain any parameters or exceptions.
* It is not specific to any object.
* It is **executed only once in the life time of every class**. I.e. it is executed in either of the following cases.
* **Case 1:** When we access any static components of the class before creating any objects for the class. (or)
* **Case 2:** when we create first object of the class before accessing any static components of the class.

**Note:**

If a class contains static and instance Constructor and if we instantiate first object before accessing any static components, than SAP first executes Static Constructor and then the Instance Constructor will be executed on behalf of that first object, from the second object onwards only instance constructor will be executed.

If an instance constructor contains any mandatory importing parameters, we must pass the values to those parameters while creating objects itself.

**INHERITANCE**

It is a process of using the properties (components) of one class inside other class.

The main aim of Inheritance is Reusability of the components i.e. a component declared in one class is accessible inside other class.

In Inheritance, two classes are involved (super class/base class **and** sub class/derived class).

The class which gives properties is called as super class / base class.

The class which takes the properties is called as sub-class/derived class.

Only Public and Protected components can be inherited i.e. Private Components cannot be inherited.

**Types of inheritance:**

1. **SINGLE INHERITANCE:** A Class acquiring properties from only one super class.
2. **MULTIPLE INHERITANCE:** A class acquiring properties from more than one entity (interface).

**Note:** In ABAP, Multiple inheritance can be implemented using the combination of class and interfaces.

1. **MULTILEVEL INHERITANCE:** A class acquiring the properties from another sub-class.

**Inheriting from** – is a keyword used as part of local class definition for inheriting another class.

A class declared as Final cannot be inherited i.e. it cannot have subclass. In case of local classes, we use the keyword ‘final’ as part of class definition to declare a final class.

**Polymorphism**

Poly -> many,

Morph-> forms,

Ism-> behaviour

It is a process of making an entity behaving in multiple forms.

In this case, the entity is a method

**Example:**

Method overloading, Method Overriding.

**Method Overloading:**

It is similar to function overloading in C++.

It is a process of overloading the same method by passing different Number and different types of parameters.

Eg: methods m1.

Methods m1 importing x type i.

Methods m1 importing x type c.

Methods m1 importing x type I y type i.

**ABAP does not support method overloading.**

**Method Overriding:**

* If a subclass redefines a super class method it is called as Method Overriding.
* Whenever a local subclass wants to redefine the super class method, the sub class as to re-declare super class method by using **“redefinition”** keyword.
* Whenever the super class method is redefined in subclasses, we cannot change the visibility / category.
* Only public and protected instance methods can be redefined.
* Whenever a subclass redefines super class method, it is always recommended to call the super class method implementation in the subclass and this is done by using “super” keyword.
* **“Super”** keyword is always used in subclass method implementations (inside redefined super class method) to call the super class method implementation.
* A class declared as final cannot be inherited.
* Static methods cannot be redefined
* Instance public / Instance protected methods declared as final can be inherited but cannot be redefined.
* A static public/static protected methods cannot be declared as final because by default, static methods cannot be redefined.

**Hierarchy of constructor execution-scenario 1**

* If a super class contains static and instance constructor and subclass without any constructors, and if we instantiate first object of super class/sub class before accessing any static components of super/sub class ,then SAP first executes static constructor of super class and then instance constructor of super class and from second object of super class/sub class only instance constructor of super class will be executed.

**Hierarchy of constructor execution-scenario 2**

* If a super class contains static and instance constructor and subclass only with static constructor, and if we instantiate first object of sub class before accessing any static components of super/sub class and also before creating any objects for super class, then SAP first executes static constructors from super class to sub class and then instance constructor of super class and from second object onwards, only instance constructor of super class will be executed.

**Hierarchy of constructor execution-scenario 3**

* If a super class contains static and instance Constructor and subclass also with static and instance constructor, then it is mandatory for sub class instance constructor to call the super class instance constructor explicitly by using super keyword.
* In this case, if we instantiate first object of sub class before accessing any static components of super/sub class and before creating any objects for super class, then SAP first executes static constructors from super class to sub class (top to bottom) and then instance constructors from sub class to super class (bottom to top) and from second object of sub class onwards, only instance constructors will be executed from sub class to super class.

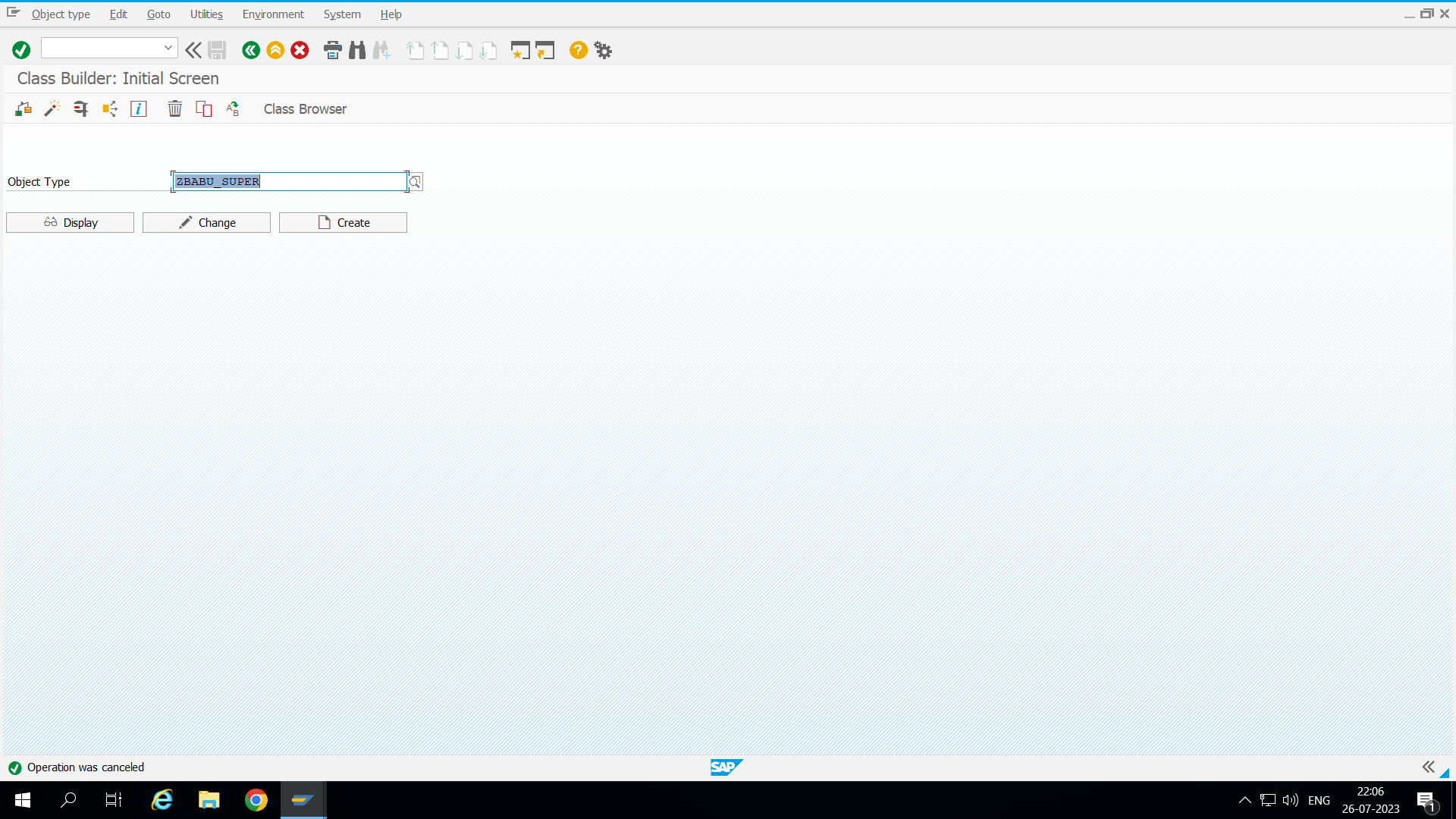
**Hierarchy of constructor execution-scenario 4**

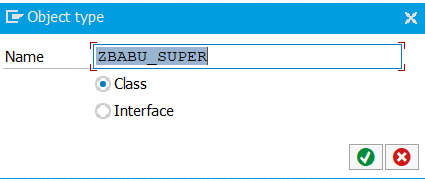
If a super class contains instance constructor with mandatory parameters and sub class also contains instance constructor with mandatory parameters and whenever we instantiate the sub class, make sure that you pass values for the parameters of sub class instance constructors and from sub class instance constructor implementation we need to call super class instance constructor by passing values for parameters of super class instance constructor.

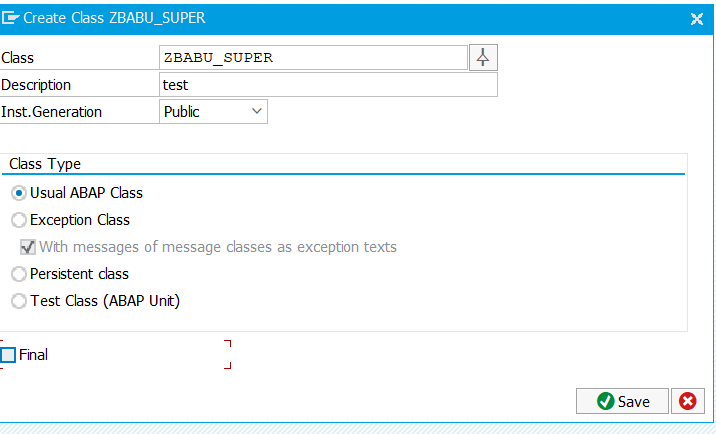
**Hierarchy of constructor execution-scenario 5**

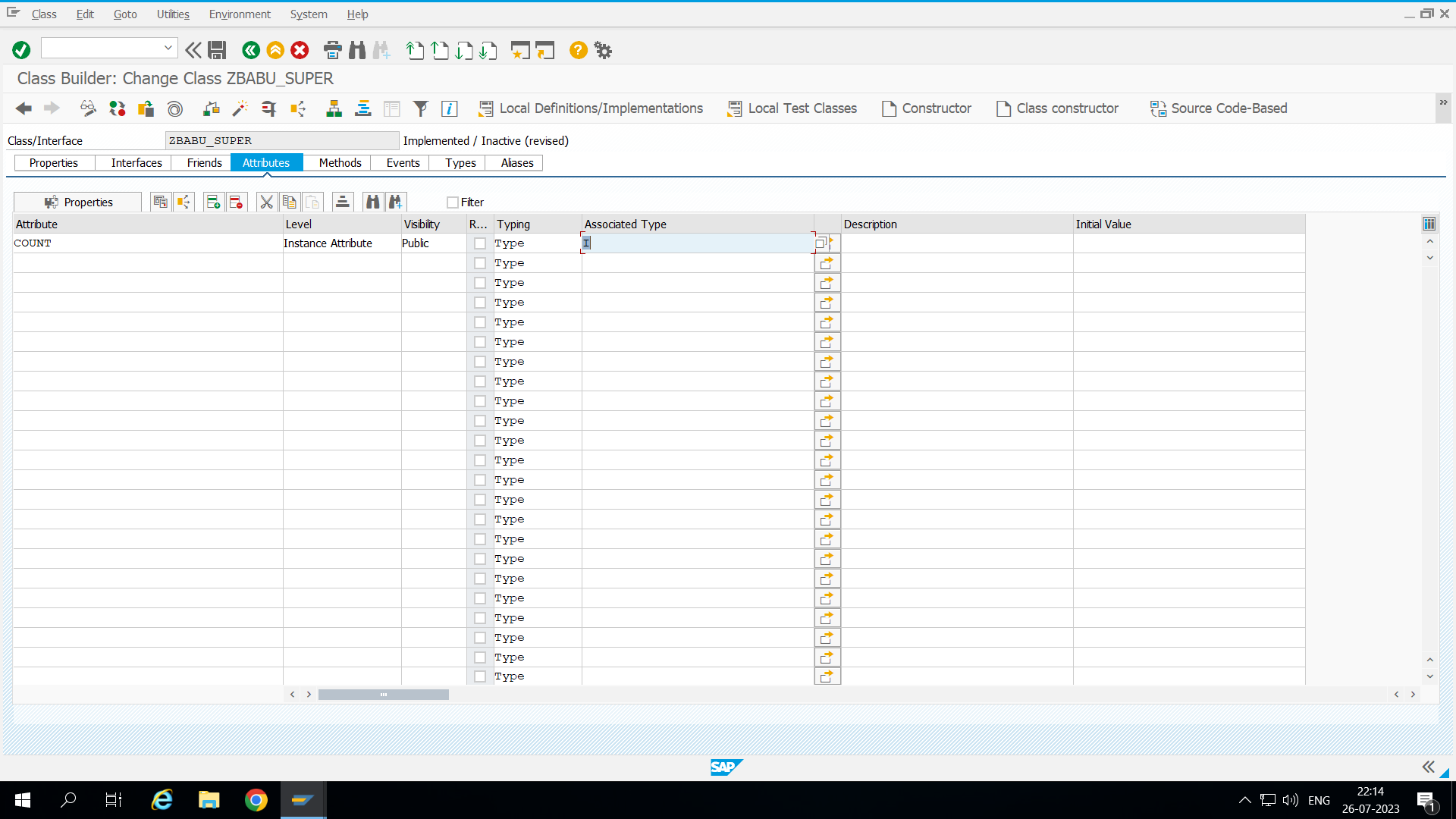
If a super class contains instance constructor with mandatory parameters and sub class without any instance constructor and whenever we instantiate the sub class, make sure that you pass values for the parameters of super class instance constructors while instantiating sub class object.

To create global class Tcode se24

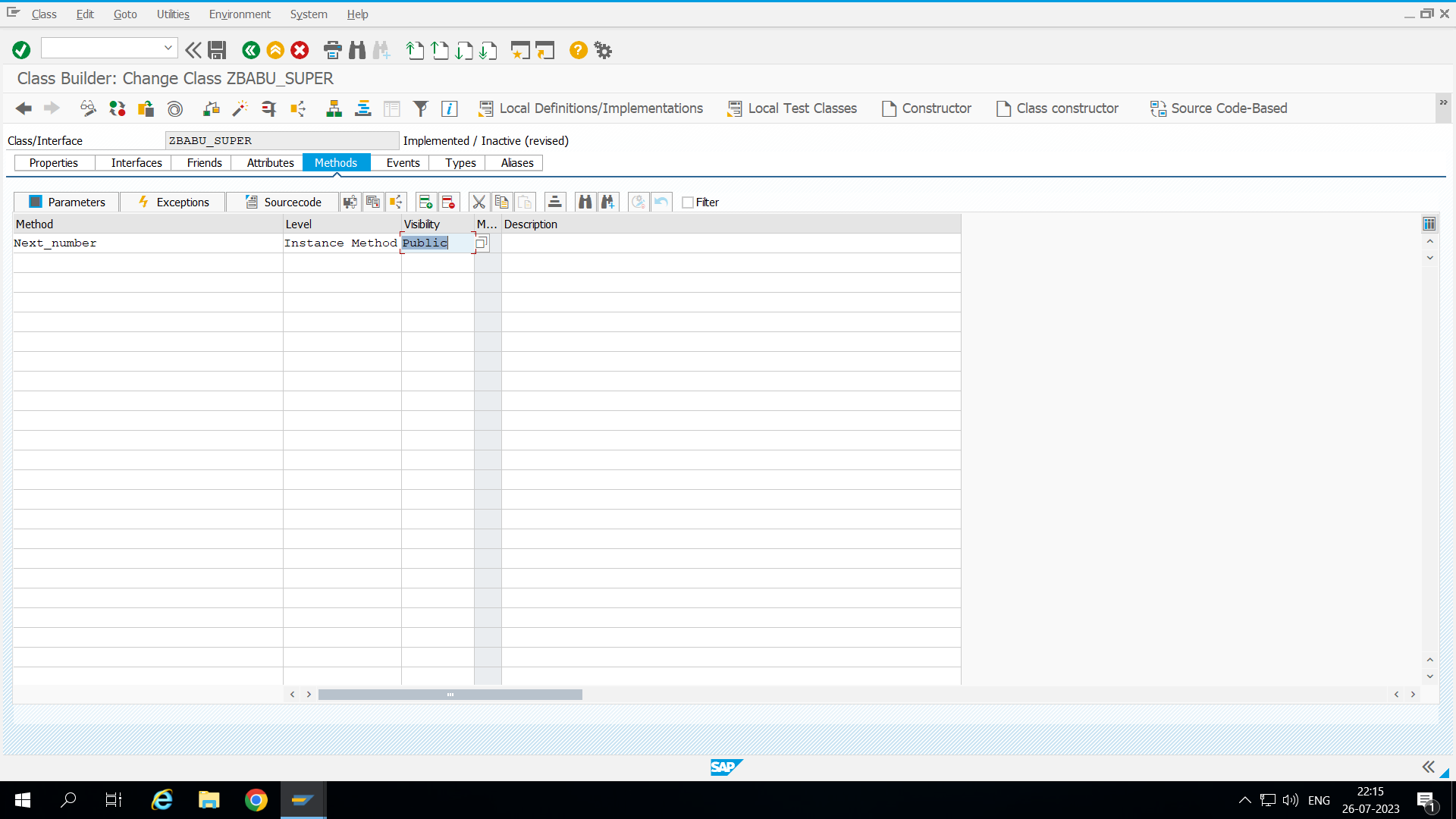


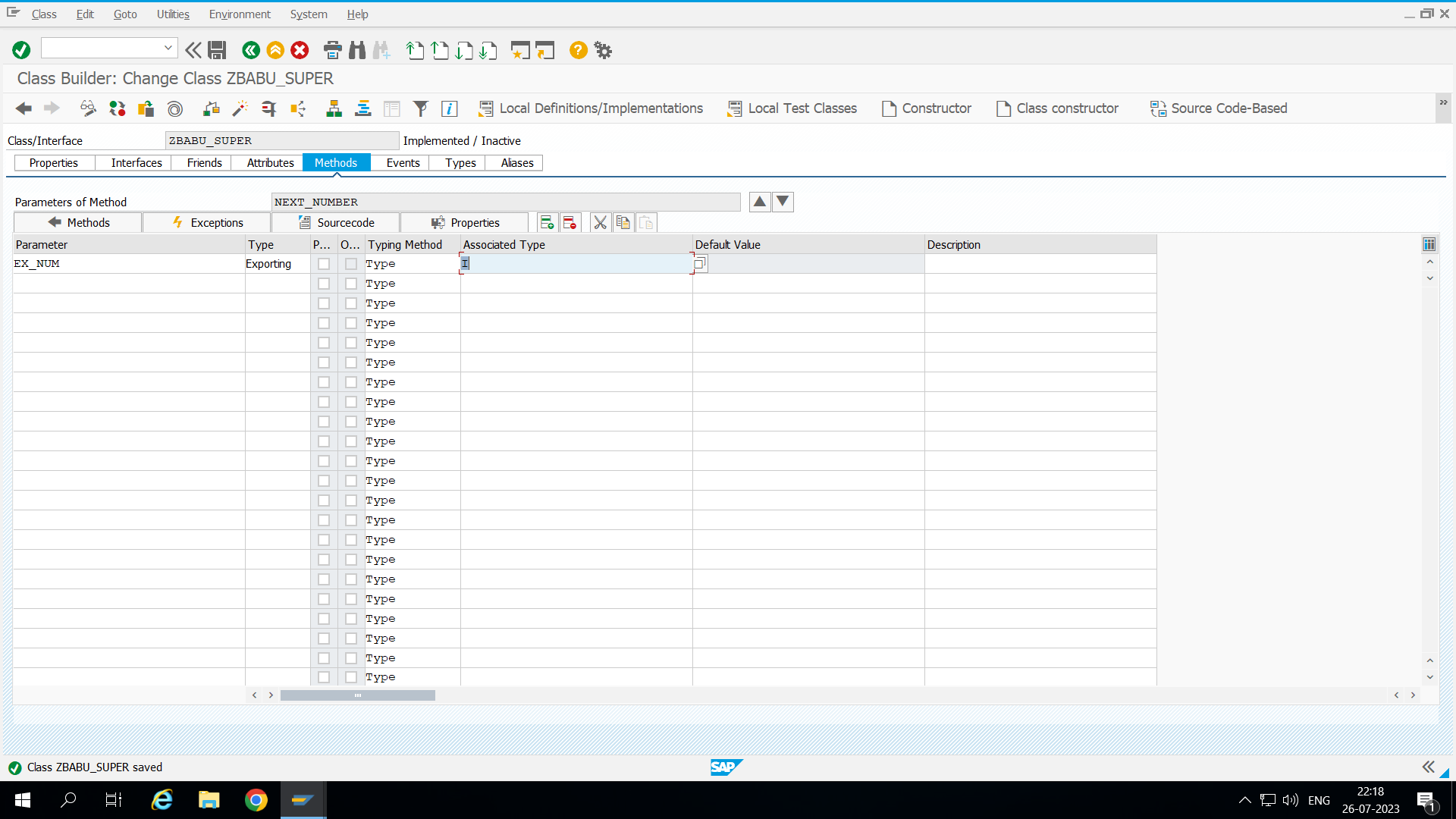


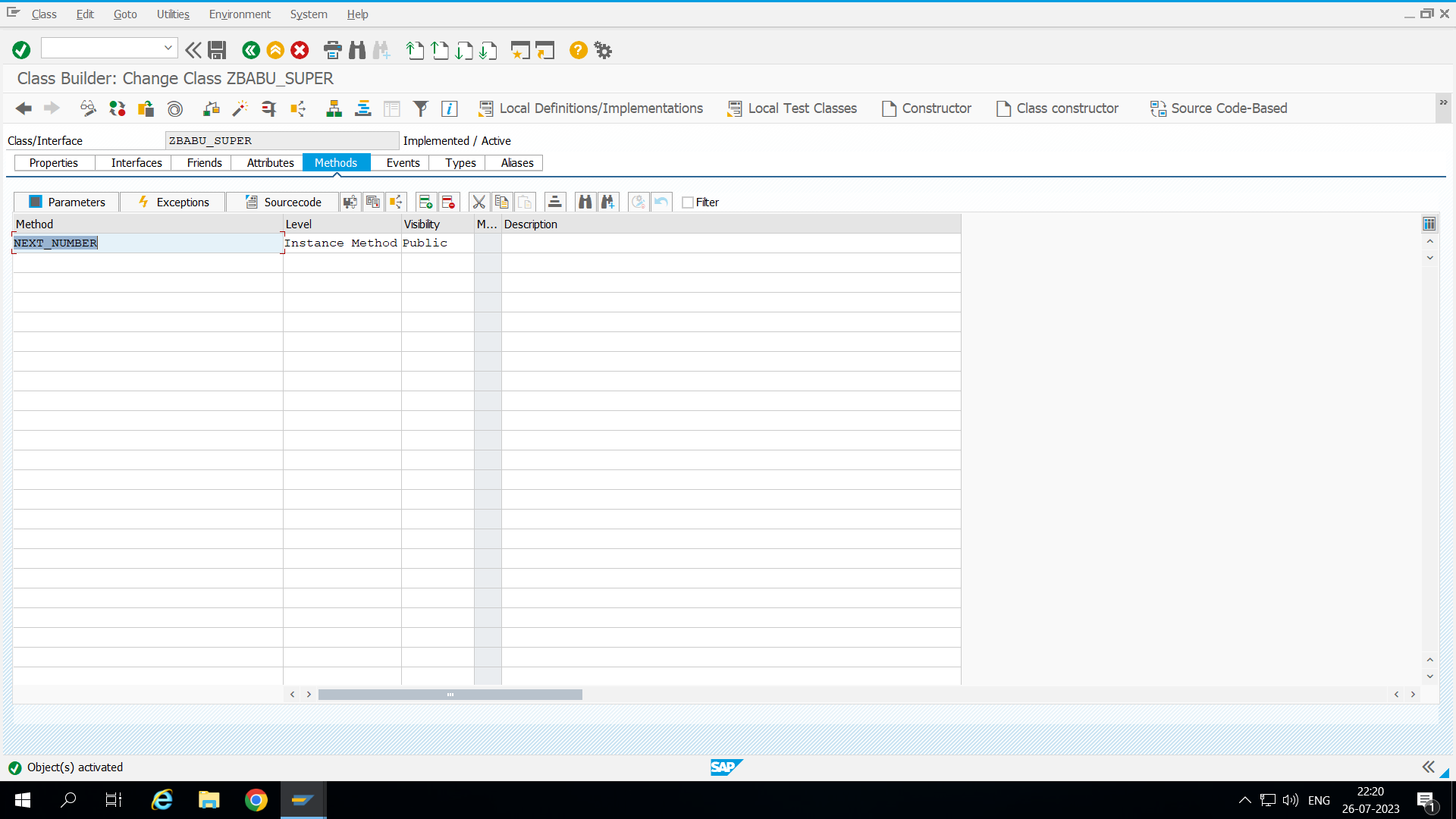




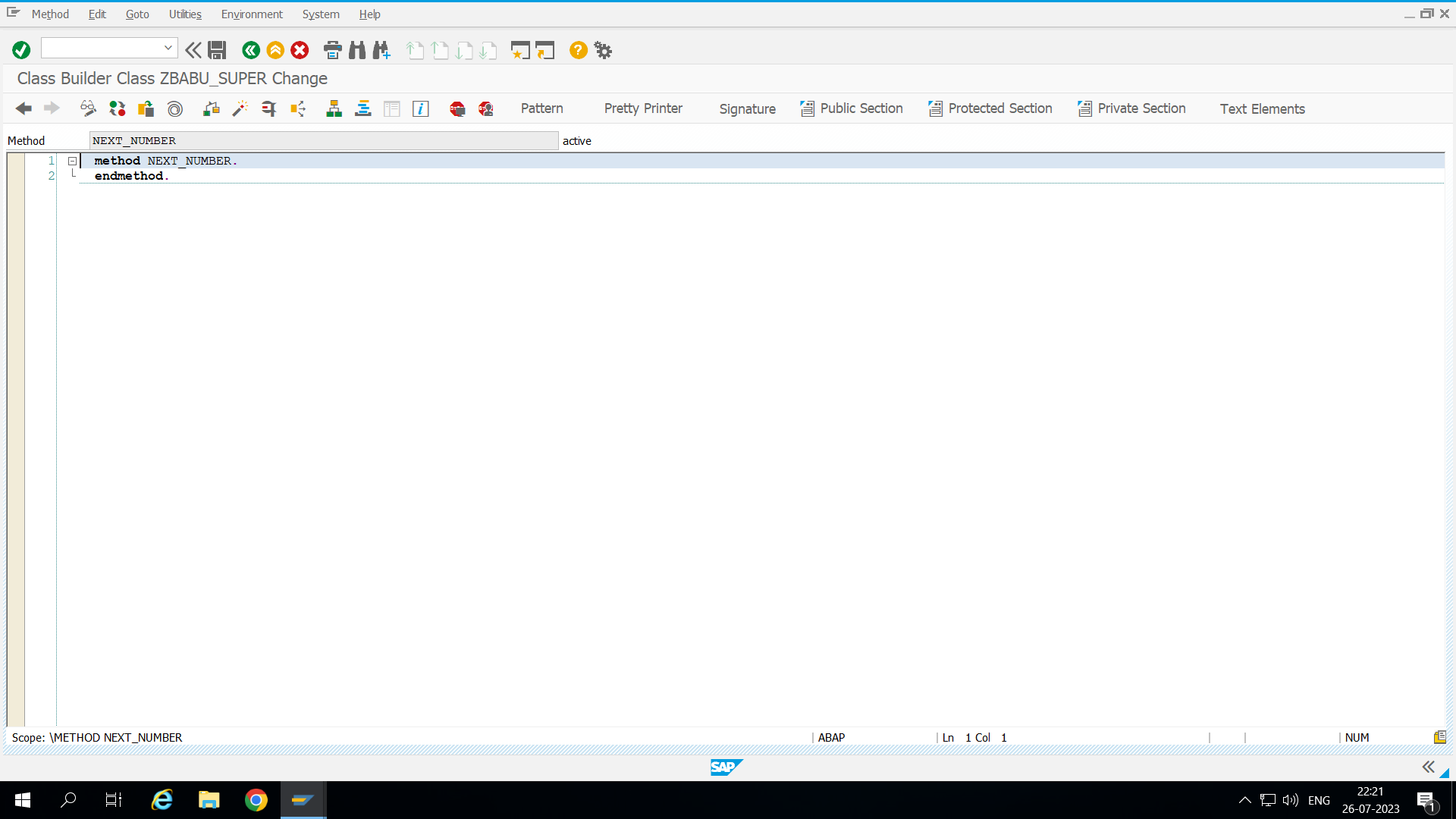
Here we taking count as Attribute count

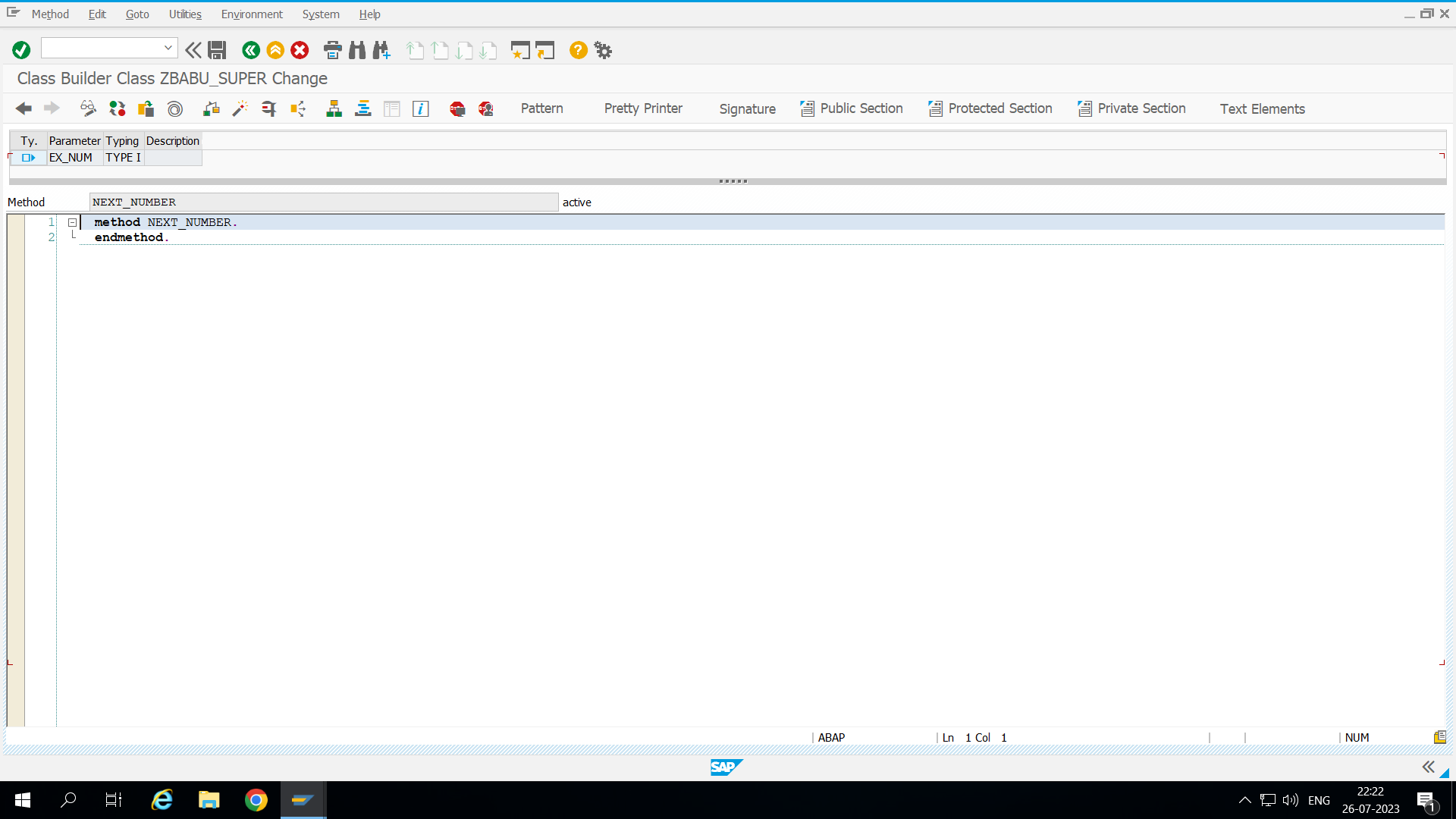
Here we defining Next\_number as METHOD to give parameter for this method place curser on the method and press parameter it’s directly take u to parameter.

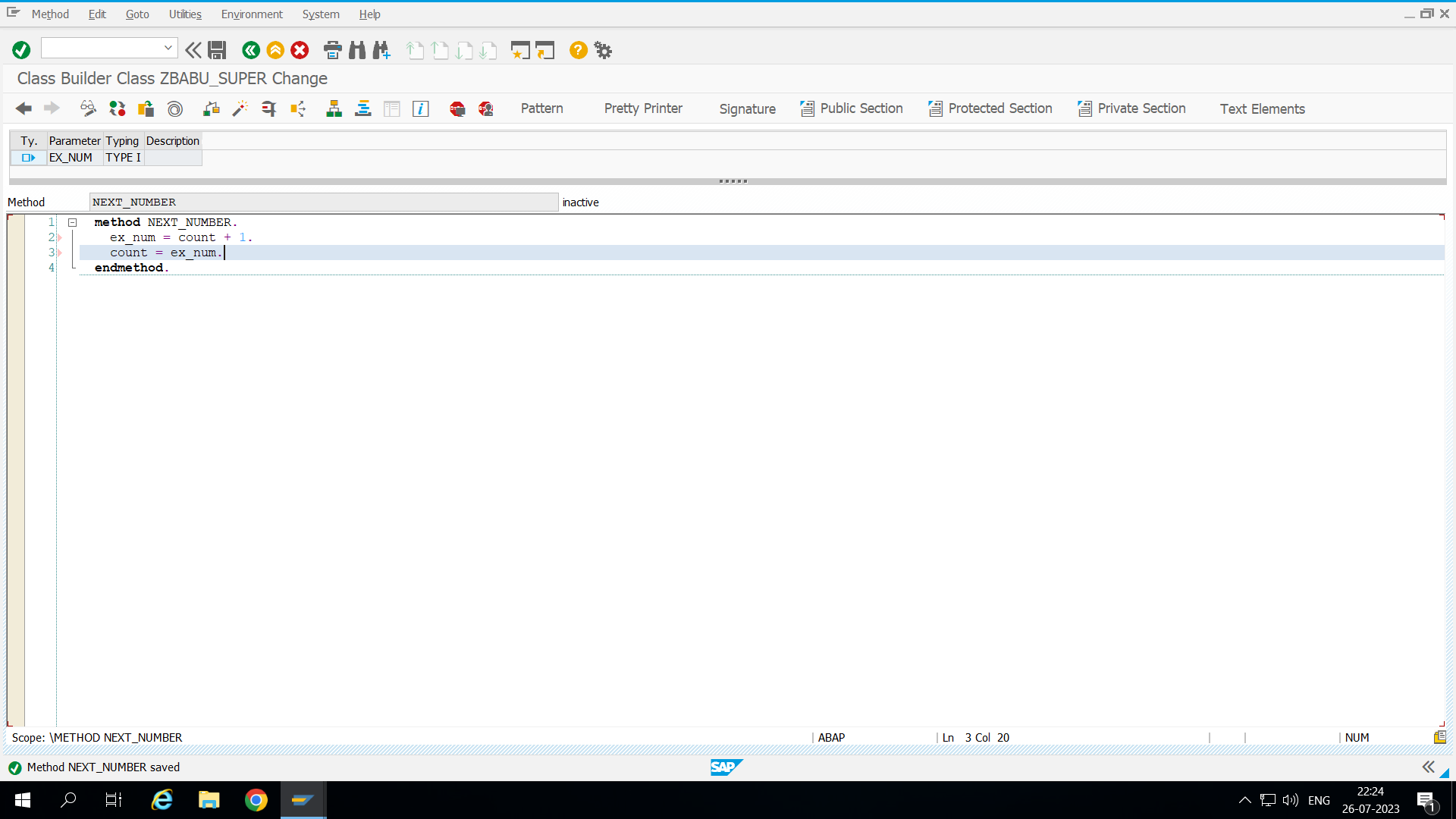


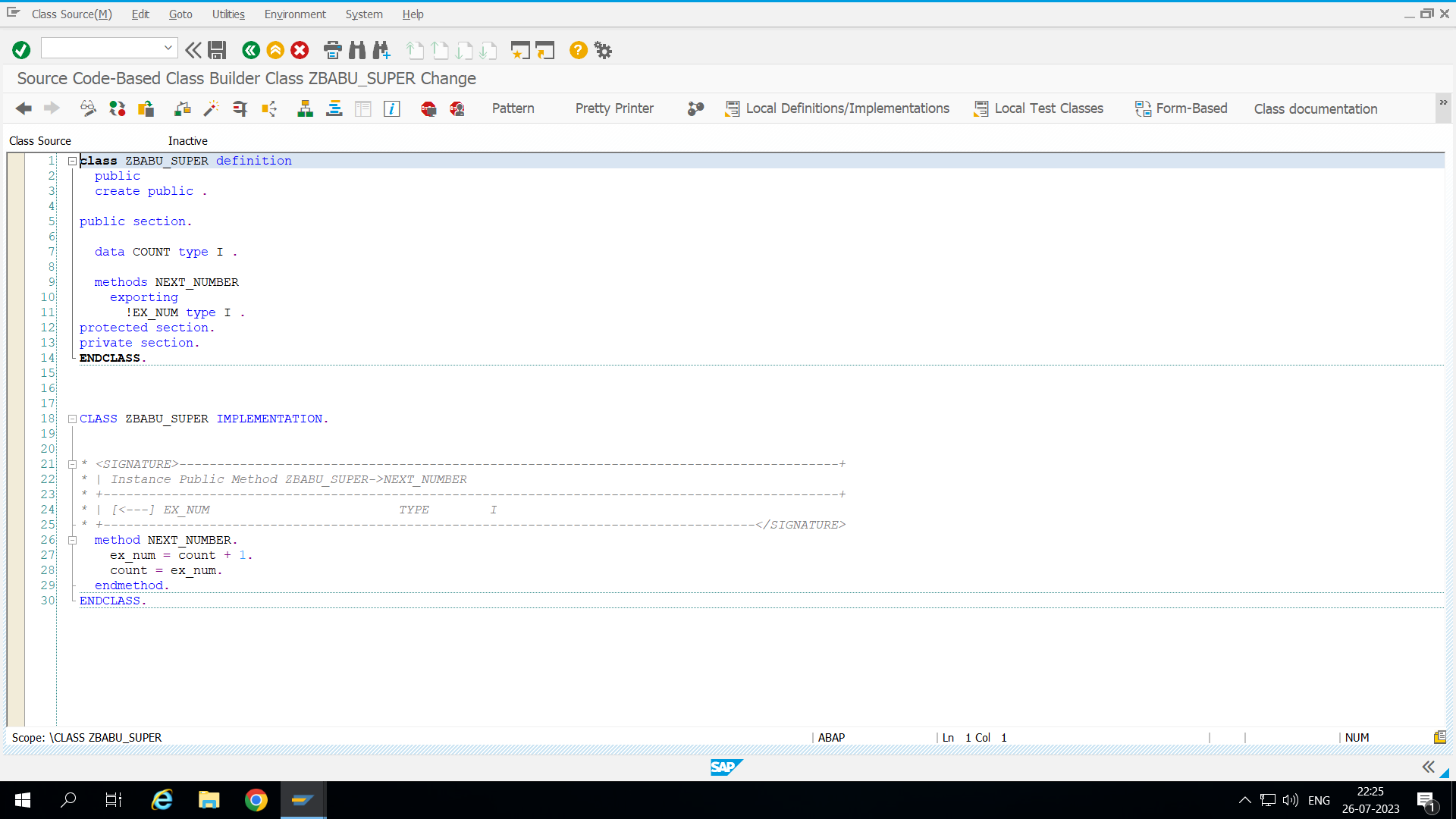


We need to activate this class and double click on method or we can go for source code to write method



If we press on signature we will get the data types.

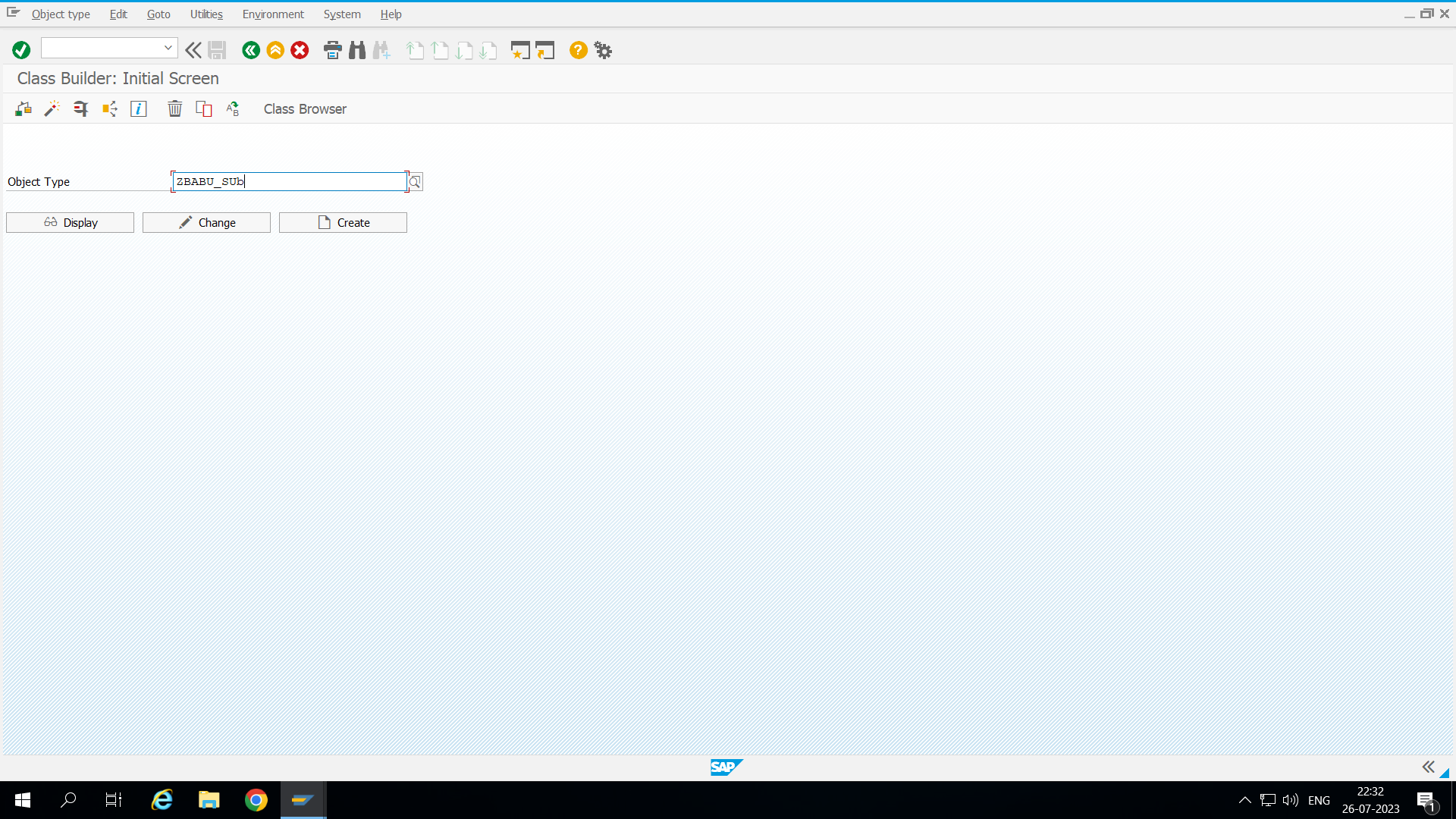


if we click on source code based the method update in that.

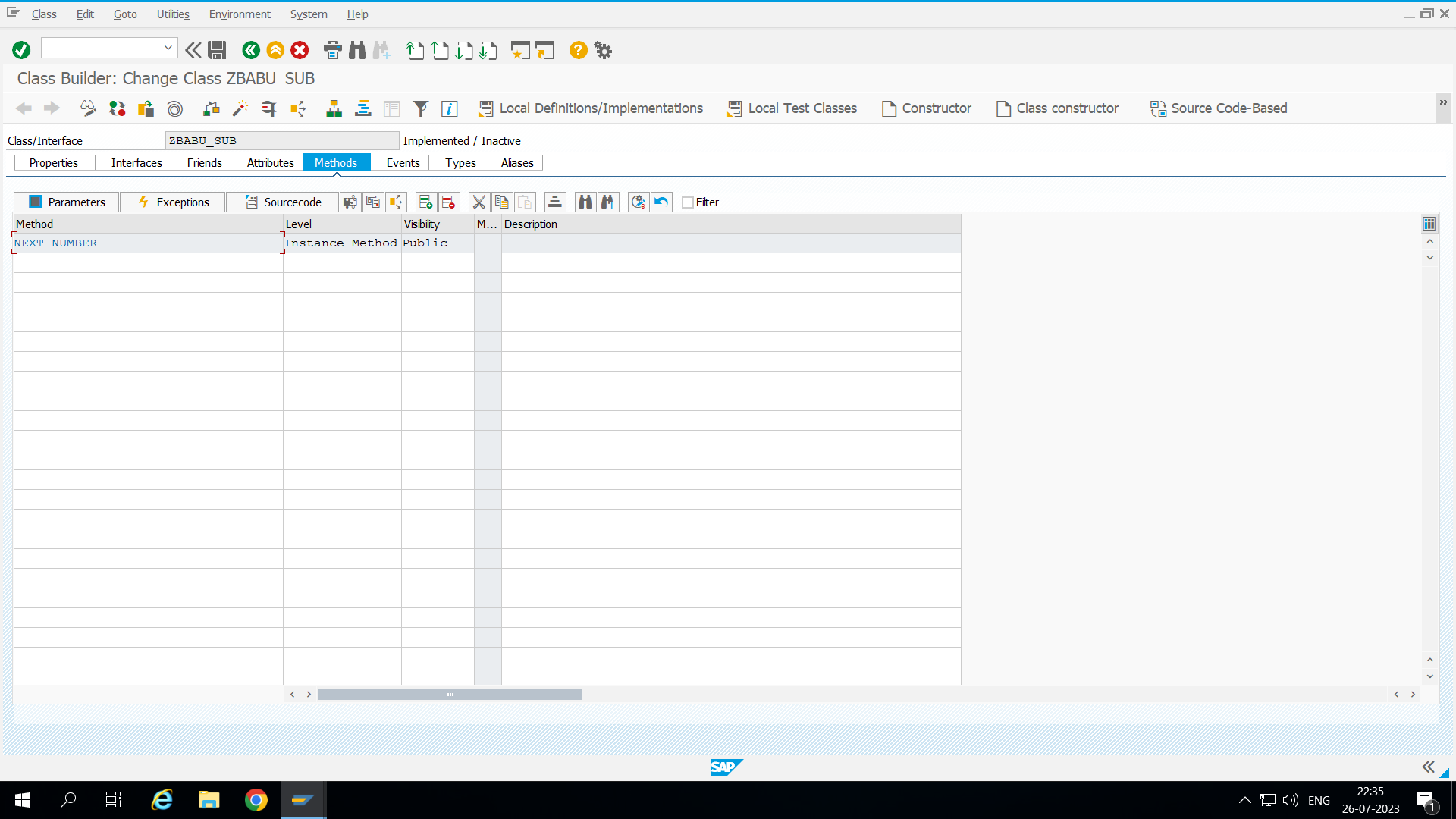
This global class can use in multiple program directly we refer to the object.

………………………………………………………………………………………………

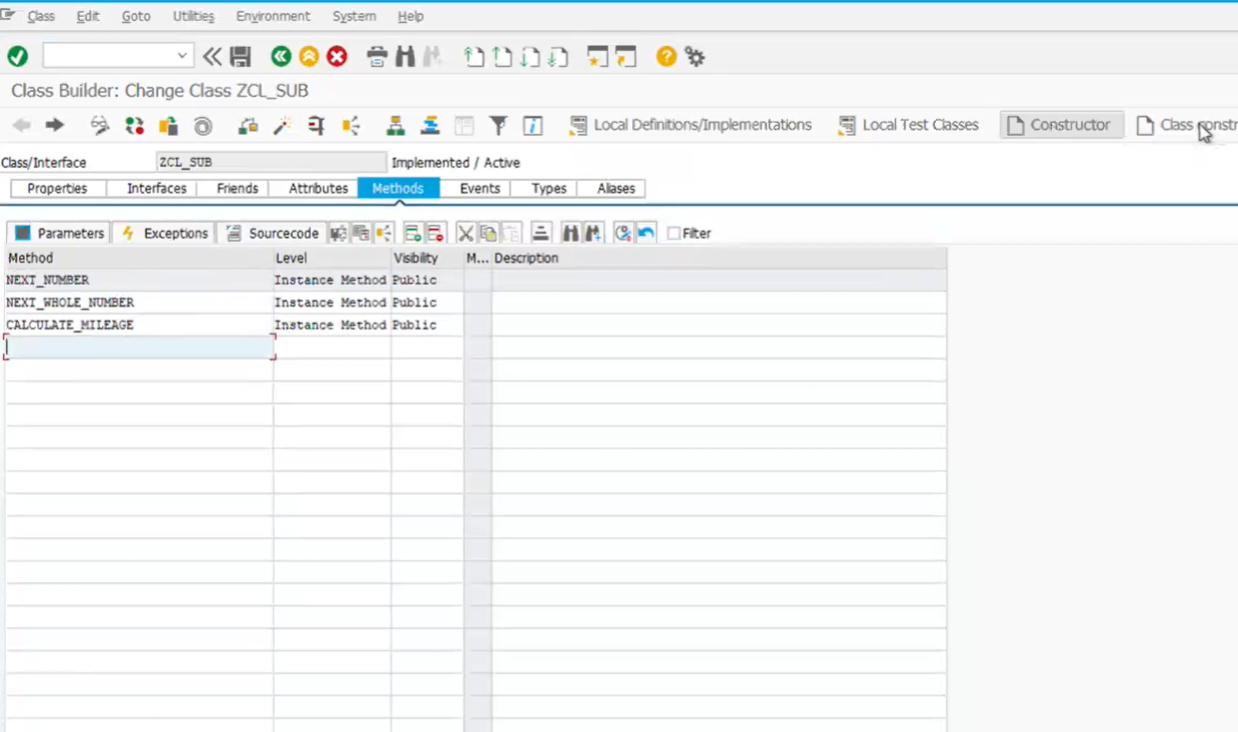
Here we creating inheritance program

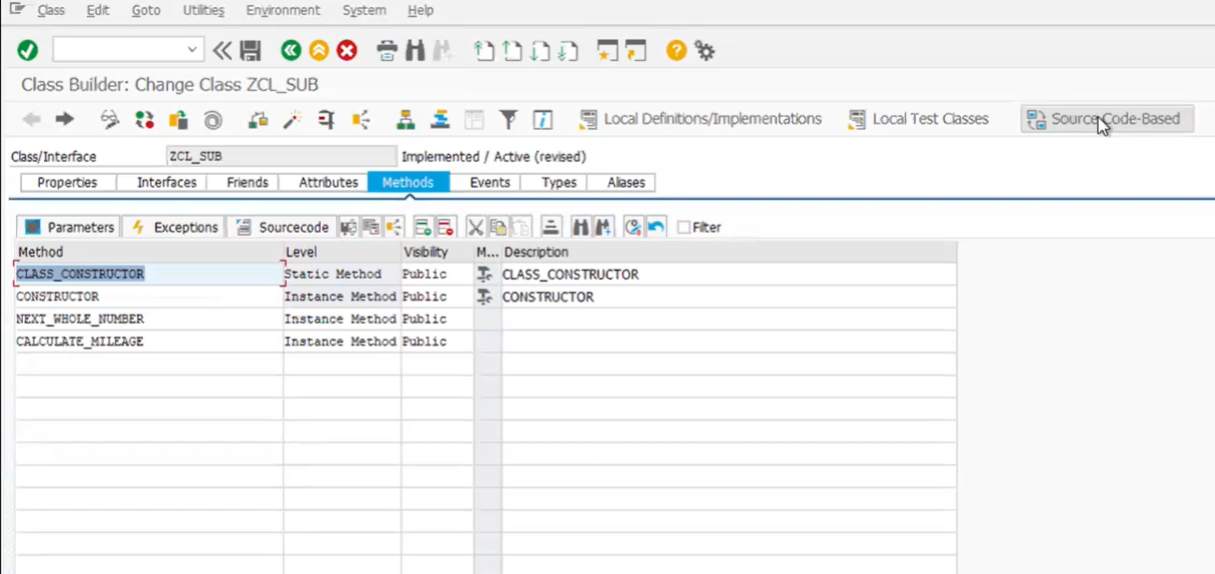


**Here we creating sub class to inherite the super class.**



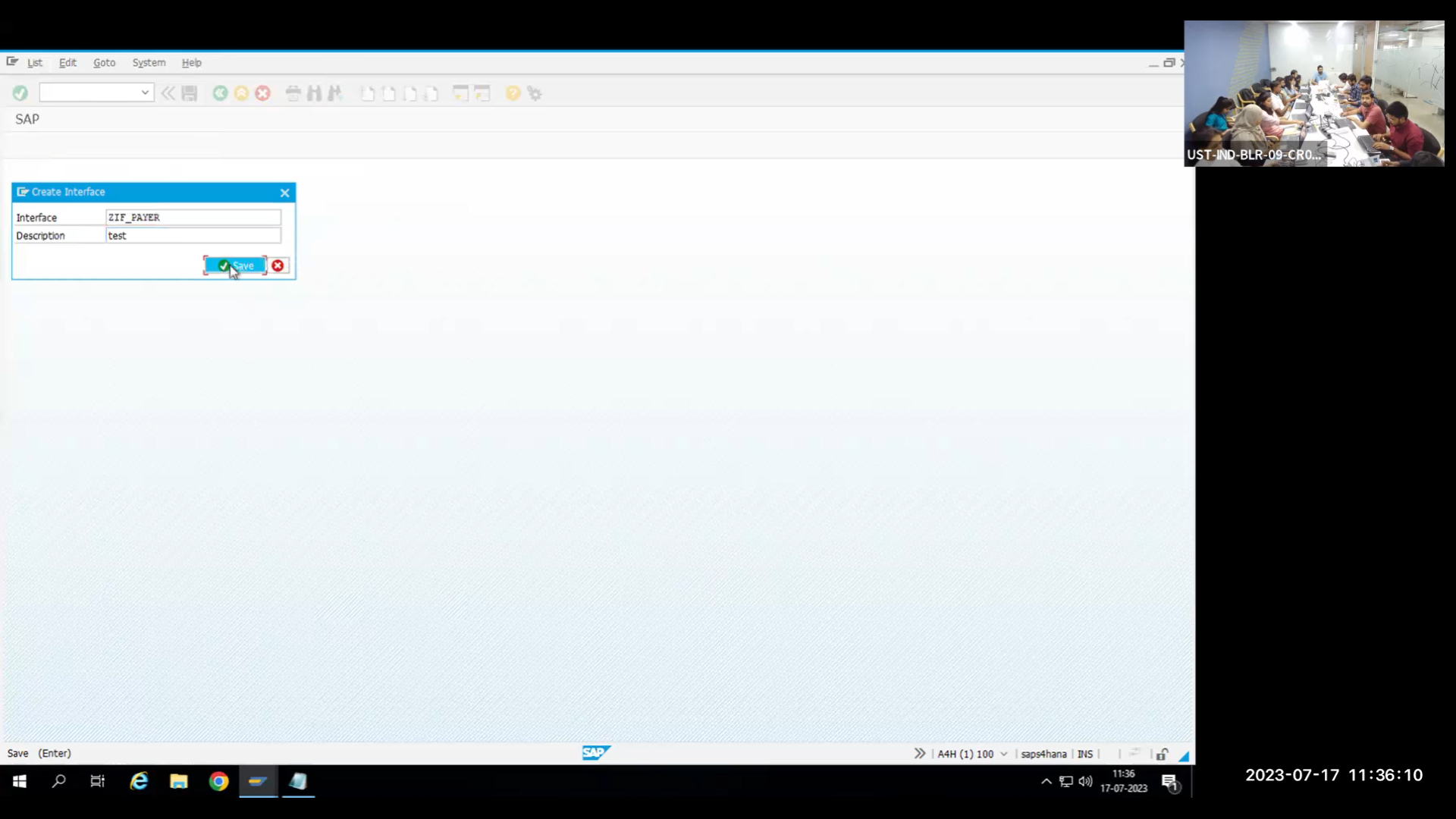
When we inherit super and sub class. This everything which we define in the super class.

if we want to create a constructor click on constructor.



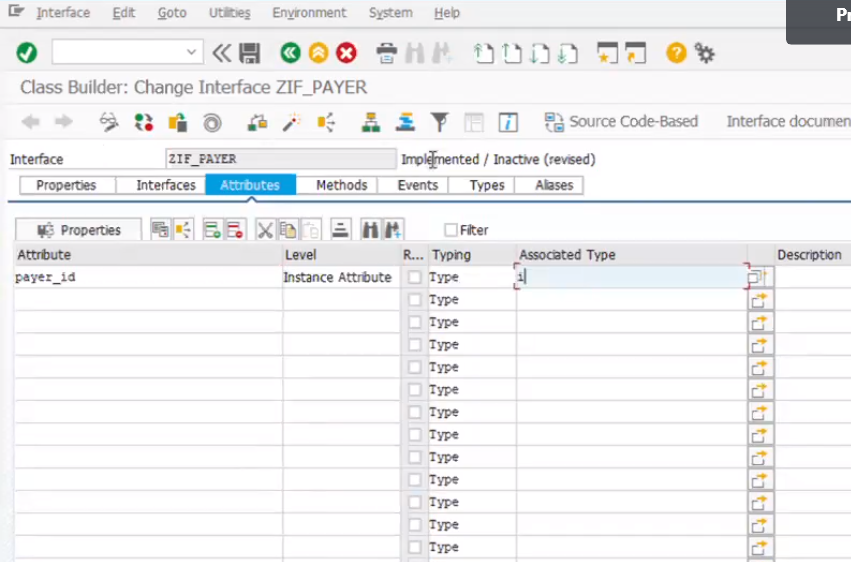
data o\_object TYPE REF TO zbabu\_super. *"Here we directly refering the globalclass*  
data o\_sub TYPE REF TO zbabu\_sub.  
  
create object:o\_object.  
CREATE OBJECT:o\_sub.  
  
o\_object->next\_number(  
 IMPORTING *"here we using instead of declaring new data type directly we declaring by using DATA()*  
 ex\_num = DATA(num)  
).  
  
o\_sub->next\_number( *"Here we declaring sub class we caling inherited method its shows the super class method*  
 IMPORTING *"One class can have one super class*  
 ex\_num = DATA(num1)  
).  
  
o\_sub->next\_wholenumber( ).  
  
  
  
o\_object = o\_sub. *"When we call this the super class pointing memory of the subclass. this dynamic while declaring the object*  
*"o\_sub ?= o\_object. "This throw syntax error both r same but we can't cast to cast sub to super we neet provide ?*  
  
o\_object->next\_number( *"At this point in my program there is no object that is pointing to the superclass and it changing of the behaviour is called polymorphism*  
 *"And this is also called casting. casting the one object to another object*  
 IMPORTING  
 ex\_num = DATA(num2)  
).  
  
write 'END'.

To create Interface.

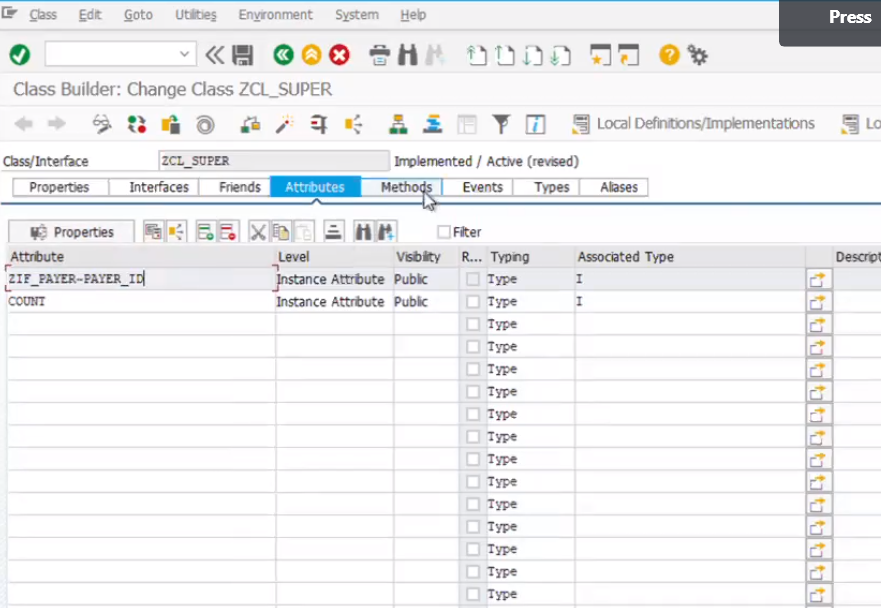


Interface can be used in any class. If you use this interface in a class, then it will derive all the definitions from the interface.

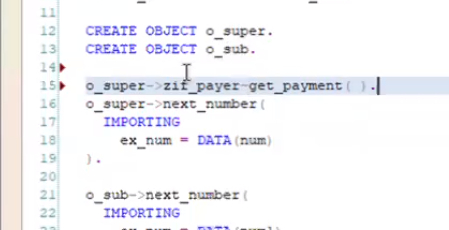




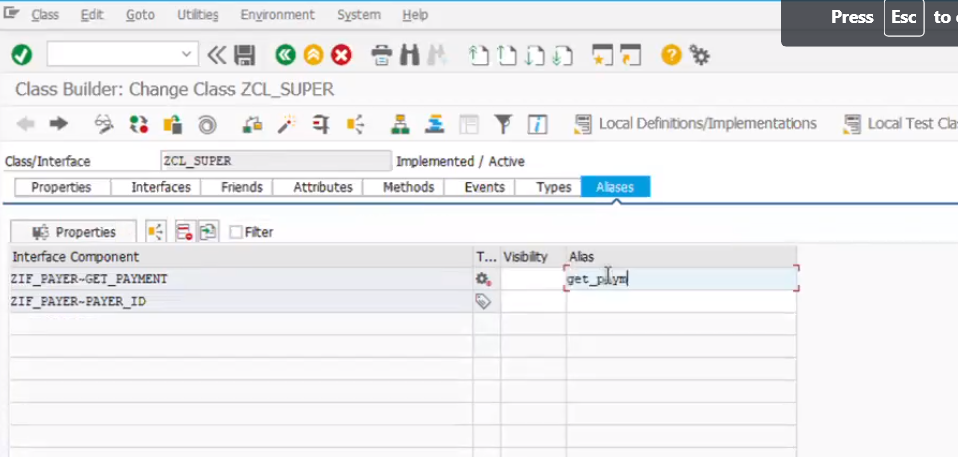
Here interface can access in any class here we referring the interface in super class

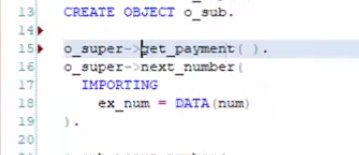


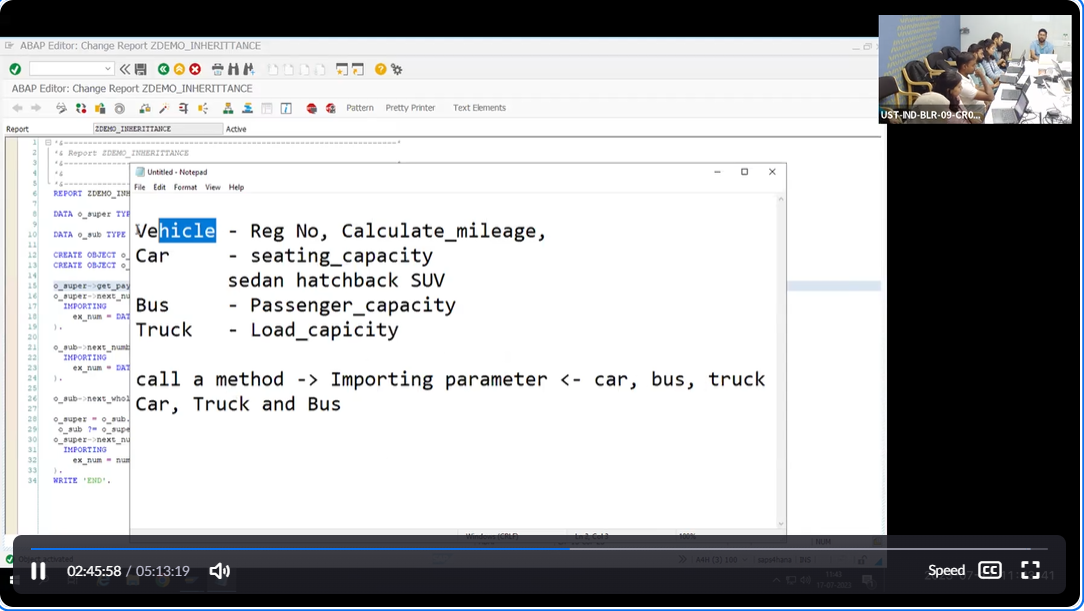
We got interface attributes in super class



We can visibility and we give alias here







Friend class can’t access private components of super class.

**Example**

DATA : it\_makt TYPE STANDARD TABLE OF makt.  
DATA v\_matnr TYPE matnr.  
  
*\*DATA o\_alv TYPE REF TO CL\_SALV\_TABLE. "To inherite this class method we are refering here*  
*\*DATA o\_alv1 TYPE REF TO CL\_SALV\_TABLE.*  
SELECT-OPTIONS s\_matnr FOR v\_matnr.  
START-OF-SELECTION.  
PERFORM get\_data.  
  
END-OF-SELECTION.  
PERFORM display.  
*\*&---------------------------------------------------------------------\**  
*\*& Form get\_data*  
*\*&---------------------------------------------------------------------\**  
*\*& text*  
*\*&---------------------------------------------------------------------\**  
*\*& --> p1 text*  
*\*& <-- p2 text*  
*\*&---------------------------------------------------------------------\**  
FORM get\_data .  
SELECT \* FROM makt INTO TABLE it\_makt WHERE matnr In s\_matnr.  
ENDFORM.  
*\*&---------------------------------------------------------------------\**  
*\*& Form display*  
*\*&---------------------------------------------------------------------\**  
*\*& text*  
*\*&---------------------------------------------------------------------\**  
*\*& --> p1 text*  
*\*& <-- p2 text*  
*\*&---------------------------------------------------------------------\**  
FORM display .  
CL\_SALV\_TABLE=>factory(  
 IMPORTING  
 r\_salv\_table = DATA(o\_alv) *" Basis Class Simple ALV Tables*  
 CHANGING  
 t\_table = it\_makt  
).  
  
DATA: lo\_functions TYPE REF TO cl\_salv\_functions\_list.   
*\**  
*\* lo\_functions = o\_alv->get\_functions( ).*  
*\* lo\_functions->set\_default( abap\_true ).*  
*\*CATCH cx\_salv\_msg. " ALV: General Error Class with Message*  
o\_alv->set\_screen\_status(  
 report = 'ZDEMO\_ALV'  
 pfstatus = 'SALV\_TABLE\_STANDARD'  
*\* set\_functions = cl\_salv\_model\_base=>c\_functions\_none*  
).  
o\_alv->display( ).  
ENDFORM.

**FORM display**: This subroutine is used to display the ALV table on the screen.

**CL\_SALV\_TABLE=>factory**: This method is used to create an instance of the **CL\_SALV\_TABLE** class, which is the basis class for simple ALV tables. It takes the reference variable **o\_alv** as an importing parameter to hold the created instance, and **it\_makt** as a changing parameter to specify the data for the ALV table.

**DATA: lo\_functions TYPE REF TO cl\_salv\_functions\_list.**: This declares a reference variable **lo\_functions** of the class **CL\_SALV\_FUNCTIONS\_LIST**, which is used to handle the ALV functions (e.g., sorting, filtering).

**o\_alv->set\_screen\_status**: This method is used to set the screen status for ALV. It specifies the ALV display settings, such as the report name and the PF-STATUS (Program Function Status) to be used.

**o\_alv->display( )**: Finally, this method is called to display the ALV table on the screen.