# OOABAP

## ( Object Oriented ABAP )

**Swapnil More**
SAP ABAP Developer @SVKM
| S/4 Hana |OO-ABAP |

**OBJECT:** Object is a runtime instance of a class.

- It is a collection of attributes (or) components of a class.
- The ABAP statement '**CREATEOBJECT'** is used to create an object

**Syntax**:

DATA:<ref_name> TYPE REF TO<class name>

CREATE OBJECT <ref_class>

Here <ref_class> is a runtime instance of a class.

**CLASS :** It is used to describe(or)define an object which can contain different attributes(or)components such as types,data,methods,constants etc.,

Class are of two types,

1. Global class
2. Local class

**1. Global Class:**The global classes are maintained in the class pool.

- The global class and the attributes can be referred across the system
- The T-code SE24 is used to work with global class.

**2. Local Class**: The local class are defined and implemented in the same program.

Each class has two sections

- Definition section
- Implementation section

a)**Classdefinition**: This section is used to define (or) declare the different attributes such as data objects, constants, methods etc.,

The ABAP statement 'CLASS.....DEFINATON' is used to define the class.

**Syntax**:

*class definition section

CLASS<class_name>DEFINTION

*visibility section

PUBLIC CLASS

Swapnil More
SAP ABAP Developer @SVKM
| S/4 Hana |OO-ABAP |

DATA:<gv_var(10)> TYPE c,

METHODS:<method_name>,

ENDCLASS.

b**) Class implementation** : It is used to implement the method of a class

- The ABAP statement 'CLASS.....IMPLEMENTATION' is used to the method of a class

**Syntax**:

*class implementation

CLASS<class_name> IMPLEMENTATION.

*implement method as a class

METHOD:<method_name>...

*get data.

SELECT

.....

ENDSELECT.

ENDMETHOD

ENDCLASS

Note:The ABAP statement 'TABLES' shouldn't be used within the classes

- The internal tables must not be declared with header line (or) occurs in the class.
- The ABAP statement 'LIKE' cannot be used to define the data objects in the classes
- The selection-screens of the report statementcannot be defined within the class.

**TYPEREFTOstatement**: this statement is used to define the different type of objects such as classes, interfaces, BADI definitions etc.,

**Syntax**: DATA : <ref_obj> TYPE REF TO <class_name/imp_name.....> .

**Different Visibility Section**:

Following are the different visibility section n OOABAP .

i. PUBLIC SECTION .
ii. PROTECTED SECTION .
iii. PRIVATE SECTION .

1. **PUBLIC SECTION Attributes**:
   - Public section attributes can be accessed in the same class, subclasses and external classes (anywhere).
   - The ABAP statement 'PUBLIC SECTION' is used to define (or) declare the public section attributes.
2. **PROTECTED SECTIONAttributes :**
   - The attributes of a protected section can be accessed in the same class and subclasses only.
   - The protected section attributes cannot be accessed in the external classes .
   - The ABAP statement 'PROTECTED SECTION' is used to define (or) declare the protected section attributes
3. **PRIVATE SECTION Attributes :**
   - The private section attributes can only be accessed in the same class.
   - The private section attributes cannot be accessed in subclasses and external classes.

   The attributes of the class should be defined in the below order

   - Public Section.
   - Protected Section
   - Private Section

| VISIBILITY SECTION | SAME CLASS | SUBCLASS | ANOTHER EXTERNAL CLASS |
|---|---|---|---|
| Public section. | YES | YES | YES |
| Protected section | YES | YES | NO |
| Private section | YES | NO | NO |

**Note:** PRIVATE SECTION attributes can be accessed in friend classes.

Syntax: CLASS <cls_friend> DEFINITION FRIENDS <lcl_name>.

ENDCLASS.


Swapnil More
SAP ABAP Developer @SVKM
| S/4 Hana |OO-ABAP |

**Different Types OfAttrbutes**:

Following are the different types of attributes.

i. Instance Attributes
ii. Static Attributes

1. **Instance Attributes:**

- They are existed for each instance of a class.
- The instance attributes can be defined using normal ABAP statements such as DATA, METHODS etc.,
- The reference of a class can only be used to access the instance attributes.
- The symbol instance connector "→" is used to access the instance attributes.

**SYNTAX:**

CLASS<class_name>DEFINITION.

* Visibility section

PUBLIC SECTION

* Define IntanceAttibuttes

DATA:<gv_var> TYPE <dataelement/....>,

**.....**

PROTECTED SECTION.

* Dedine Instance Methods

METHODS: <METH_NAME> IMPORTING

EXPORTING

PRIVATE SECTION." with in the class only

DATA:.....

**.....**

ENDCLASS.

* CLASS IMPLEMENTATION SECTION

**.....**

Swapnil More
SAP ABAP Developer @SVKM
| S/4 Hana |OO-ABAP |

START OF SELECTION

* Reference

DATA: <REF_CLASS> TYPE REF TO <CLASS_NAME>.

* CREATE OBJECT

CREATE OBJECT<REF_CLASS>.

*Access Instance Attributes

**syntax**:

CALL METHOD <REF_CLASS →<METH_NAME>.

2. **Static Attributes :**

- They are existed only once but can be used for any number of instances.
- The ABAP statement “CLASS --” is used to prefix to plain ABAP data declaration (or) definitions statements to define static attributes.
- The class name should only be used to access the static attributes but not class reference.
- The static attributes connector symbol (===>) is used to refer the static attributes.

**syntax:**

* Define static attributes

**CLASS-**DATA:<gv_static>....

**CLASS-**METHODS: <static_meth>....

* Access static attributes

CALL METHOD <class_name>**==>**<static _meth>

Here <class_name> is class name.

**METHOD:**

- It is a collection of statements (process blocks).
- All the attributes of a class can be accessed in the methods.
- The methods can be compared with function modules.
- ABAP statements “METHODS (or) CLASS-METHODS “are used to define the methods.
- The ABAP statement "METHOD.....END METHOD" is used to implement the methods in an implementation of a class.

**syntax:** Definition section

**.....**

PRIVATE SECTION/PUBLIC SECTION.
METHODS:<meth_name> IMPORTING<im_param>.....,
EXPORTING<ex_param>.....,
CHANGING<ch_param>.....,
RETURNING<ref_param>....,
EXCEPTIONS<ex_param>....,
*Implementation section of a class.
**.....**

METHOD<meth_name>.
*Process Block
.
.
.
SELECT .....
.
.
.
ENDMETHOD.

**NOTE:** The function method have only one return parameter without any importing and exporting parameters.

**CALL METHOD Statement:-**

- This statement is used to call the methods of the class.

**syntax:**

CALL METHOD<class_name>----> (or) ===><meth_name>

IMPORTING<Im_param>...,

EXPORTING<Ex_param>...,

**CONSTRUCTOR and CLASS_CONSTRUCTOR:**

- They are the special methods and named as CONSTRUCTOR (Instance) and CLASS_CONSRUCTOR (Static).
- Constructors are called at run time automatically while creating an object.

- The constructors should be defined under PUBLIC SECTION only.
- The instance constructor can have only Importing and exceptions but not Exporting parameters.
- The static constructor cannot have any Importing and Exception parameters.

**syntax:**

```
   * Instance constructor
  METHODS: CONSTRUCTOR IMPORTING……., .
*Static constructor
            CLASS_CONSTRUCTOR,
                    .
                    .
                    .
```

**EVENTS IN OOABAP:**

- Event in OOABAP is used to raise the messages by an object.
- The ABAP statement " EVENTS" is used to define the events.
- The ABAP statements " RAISE" is used to raise an event.

**SYNTAX:**-

```
CLASS<class_name> DEFINITION.

            .

            .

            .

EVENTS : ,event_name>.

ENDCLASS.

            .

            .

IF    SY-SUBRC  NE 0.

* Raise Event.

      Raise<Event_name>.

          .

          .
```

Swapnil More
SAP ABAP Developer @SVKM
| S/4 Hana |OO-ABAP |

ENDIF.

**Event Handler:**

- It is used to call method of one class in another class to make use of same functionality.
- The additional statement" FOR EVENT" is used to define the methods with event.
- The ABAP statement " SET HANDLER" is used to access attributes of event handler.

**SYNTAX:-**

CLASS<cl_event_handler> DEFINITION.

PUBLIC SECTION.

METHODS<event_method> FOR EVENT <event_name> OF <class_name>.

.

.

ENDCLASS.

.

.

* IMPLEMENT <cl_event_handler>.

CLASA<cl_event_handler> IMPLEMENTATION.

METHOD<event_handler>.

*Bussiness logic

ENDMETHOD.

ENDCLASS.

*Call event handler methods

SET HANDLER <ref_event> --><event_method> FOR <ref_class_name>.

**TRY....CATCH...statements:**

- This statements are used to try the specified bussiness logic (or) methods and catch the specified messages in case of fail.

**SYNTAX:**

TRY

CALL METHOD.....

CATCH <message>.

.

.

ENDTRY.

**Properties OfOoabap:**

Following are useful properties of OOABAP.

1. Abstraction.
2. Encapsulation.
3. Inheritation.
4. Polymorphism
5. Interface.

1**.Abstraction:**InOoabap Each And Everything Is Visualized As Objects Such Are

Class, objects,etc.

2.**Encapsulation:** This PropertyIs Used To Combine The Different Components Into A

Single unit (Binding) called class.

3.**Inheritance:**

- This property is used to adopt the properties of a parent class to the child classes (subclasses).
- The ABAP statement “INHERITING FROM" is used to define the subclass by adopting the properties of parent class.

**Syntax to Define Subclasses:**

CLASS<class_name> DEFINITION....

.

.

.

ENDCLASS.

* Define Subclass using Inheritance

CLASS<sub_class> INHERTING FROM <class_name>

.

.

ENDCLASS.

**NOTE:** The final classes cannot be inherited.

The final method cannot be redefined.

**Syntax To Define Final Class And Method:**

CLASS<final_class> DEFINITION FINAL.

.

.

.

METHODS:<final_meth>.....FINAL.

**NOTE:** Private section attributes can be accessed in Friends class.

**Syntax To Define Friends Classes:**

CLASS<frind_class> DEFINITION FRIENDS<class_name>

.

.

.

ENDCLASS.

4.**Polymorphism:**

- Different behaviour of same method in different classes is called polymorphism.
- The ABAP statement “REDEFINITION " is used to redefine the methods (or) classes.

**SYNTAX:**

* SUBCLASS

METHOD: <meth_name>.....REDEFINITION.

5. **Interfaces InOOABAP:**

- They are the independent structures which can be used to extend the scope of the classes
- The ABAP statement “INTERFACE" is used to define the interfaces.
- The interfaces do not have their own implementation sections.
- The ABAP statement “INTERFACES" is used to access the interfaces in a class.

**NOTE:**The interfaces must be specified under Public section of the class.

- The symbol "~ (tilt)" is used to refer the attributes of the interfaces.

**Syntax:**

*Define interfaces

INTERFACE<int_name>.

METHODS:<inter_meth>

.

.

.

ENDINTERFACE.

* Specify Interface

CLASS<class_name> DEFINITION.

.

.

.

PUBLIC SECTION.

INTERFACES:<int_name>  "must be under public section.

ENDCLASS.

* Implement interface methods

CLASS <class_name> IMPLEMENTATION.

METHOD<int_name> ~ <int_meth>.....

.

ENDMETHOD.

ENDCLASS.

**Up Casting:**

- It is also called "NARROW CAST".
- It is used to assign (or) copy instance of subclass into super class.

Eg:-<cl_superclass> =<cl_subclass>.

.

.

.

CALL METHOD <cl_superclass> ----><meth_redefine> ." calling from subclass

**NOTE:** It is not possible to access the sublcass own methods incase of upcasting.

**Down Casting:**

- It is also called " WIDENING CAST"
- It is used to assign (or)copy instance of superclass to subclass.
- The widening cast operator "?=" is used to assign (or) copy instance of superclass.

Eg: <cl_Psublcass> ?= <cl_superclass>.

**NOTE:**Incase of Down casting the subclass own methods can also be accesed.

- Create one new instance to access the attributes of a super class after up casting.

**GLOBAL CLASS:-**

**Steps to create a Global Class:-**

- Execute the t-code "SE24".
- Enter the global class name.
- click on create button.
- Select a radio button interface to create the gobal interfaces.
- Click on yes button.
- Enter short description.

**NOTE:**

1. Use the button create inheritance to create the subclass by inherting the existing super class.
2. The default class is final class.
3. Uncheck the checkbox "FINAL" to create normal global class.
4. In case of persistent class no need to create an object and the sysytem generates the defaults class.

- Click on yes button.
- Enter the package name.
- Click on save button.
- Create the "TR" by pressing an icon create request.
- Define the required methods.

EG:

| METHOD | LEVEL | VISIBILITY |
|---|---|---|
| Inst_meth | Instance method | Public |
| Static_meth | Static method | public |

- Click on save icon.
- Place a cursor on method.
- Click on parameters button.
- Define the required parameters.

**NOTE:-**Follow the similar steps to define the parameters for any number of methods.

- Click on methods button.
- Double click on method name.
- Click on yes button.
- Click on signature button to display all the parameters of method.
- implement the required business logic within the system generated statements

**Swapnil More**
SAP ABAP Developer @SVKM
| S/4 Hana |OO-ABAP |

- METHOD.....ENDMETHOD.

**NOTE:** Attributes of same classcan be referred using self reference statement "me".

- Click on pretty printer button.
- Click on save button.
- Click on back.

**NOTE:-** Follow the similar steps to implement number as methods.

- Click on activate icon.
- Press enter.

**STEPS TO CALL THE METHODS FROM THE GLOBAL CLASS (OR) INTERFACE:**

- Once the program is in change mode.
- place the cursor where the method to be called from the gobal class(or)interface
- Click on pattern button.
- Select the radio button ABAP OBJECT pattern.
- Click on yes button.
- Enter class (or) interface (or) instance name.
- Enter (OR) Select the required method.
- Click on yes.
- Uncomment the required parameters and pass the data variables from the calling program.
- Create the self reference statement "me" with the reference object name.

EG:

```
DATA:<cl_ref> TYPE REF TO <global_class>.

                    .

                    .

                    .

CREATE  OBJECt<cl_ref>.

                  .

                  .

*Call method ME ---->

CALL METHOD <cl_ref> ------><inst_meth>
```

Swapnil More
SAP ABAP Developer @SVKM
| S/4 Hana |OO-ABAP |

**NOTE:**

- The data types of the method parameters and data types by passing parameters variables should be similar. Otherwise program goes to short dump.

**USES OF OOABAP:-**

- Better programming structure.
- Abstraction and Encapsulation.
- Polymorphism and Inheritance.
- Stress on data security and data acess.

**DIFFERENCE BETWEEN LOCAL AND GLOBAL CLASS**

| LOCAL CLASS | GLOBAL CLASS |
|---|---|
| Local classes can be accessed within the same program. | It can be accessed across the system. |
| It can be started with any letter. | It should start with Z*/Y* in case of custom. |
| It can be defined and implemented in any source code. | It must be defined and implemented using t-code se24. |

**OOALV (Object Oriented ALV) REPORTS**

1. The system provided standard classes which start with "CL_GUI" are used in OOALV reports.

EG:

CL_GUL_ALV* --- " Display data in Alv Custom control.

CL_GUL_CUSTOM*/DOCKING.....

2. The system provided standard table types and structures with start with "LVC*" are used for different types which can be used for OOALV reports.

**NOTE:**

- The standard type group "SLIS" is used in case normal ALV reports.

3. The custom control in screen painter to be created to display the data in OOALV reports.

**SOME OF THE USEFULL STANDARD CLASSES TO DISPLAY AND FORMAT:-**

- **CL_GUL_ALV_GRID:-** It is used to work with ALV GRID formats in OOALV.
- **CL_GUL_ALV_TREE:-** This standard class and corresponding methods are used to work with ALV tree control (OR) tree structure.

**NOTE:**

- In the real time the standard system provided class "CL_GUL_ALV_GRID" is mostly used to work with grid format.

**SOME OF THE USEFUL METHODS OF ALV CLASSES:-**

- **DOUBLE_CLICK:-** This event handler method is used to provide the double click functionality in OOALV reports.
- **HOTSPOT_CLICK:-**This event method is used to provide double click functionality based on the hand symbol click.
- **TOP_OF_PAGE:-**It is used to populate and display header data in OOALV reports.
- **MENU_BUTTON:** It is used to call the menu status to define the pushbuttons in OOALV.
- **USER_COMMAND:-** It is used to provide an interactive functionality based on the performed active.
- **SET_TABLE_FOR_FIRST_DISPLAY:** This method is used to display the final data in grid format.

**NOTE:** It can be compared with the function module "REUSE _ALV_GRID_DISPLAY".

**DIFFERENT TYPES OF CONTAINERS:**

Following are different types of containers

1. **Custom container:** It is the normal container to display single output on custom container.
   - The standard class " CL_GUI_CUSTOM_CONTAINER" is used to work with custom container to display the data.
2. **Docking container:** It is used to display the data on custom controlled at any position either left (or) right.
   - The standard class “CL_GUI_DOCKING_CONTAINER" is used to work with custom container to display the data.

3. **Splitting container:** It is used to split the container into different parts to display the multiple grids in OOALV.

- I t is not possible to display the data in any position (Docking) in normal ALV reports.
- It is not possible to display multiple grids in normal ALV reports

- The standard class “CL_GUI_SPLITER_CONTAINER" OR "CL_GUI_EASY_SPLITER_CONTAINER". Is used to work with Splitter Container.

**SOME OF THE USEFULL TABLE TYPES AND STRUCTURES ON OOALVS**

**1.LVC_T_FCAT:** This table type can be used for filed catalogue for OOALV.

**NOTE:** It can be compared with the type “SLIS_T_FILEDCAT_ALV" .

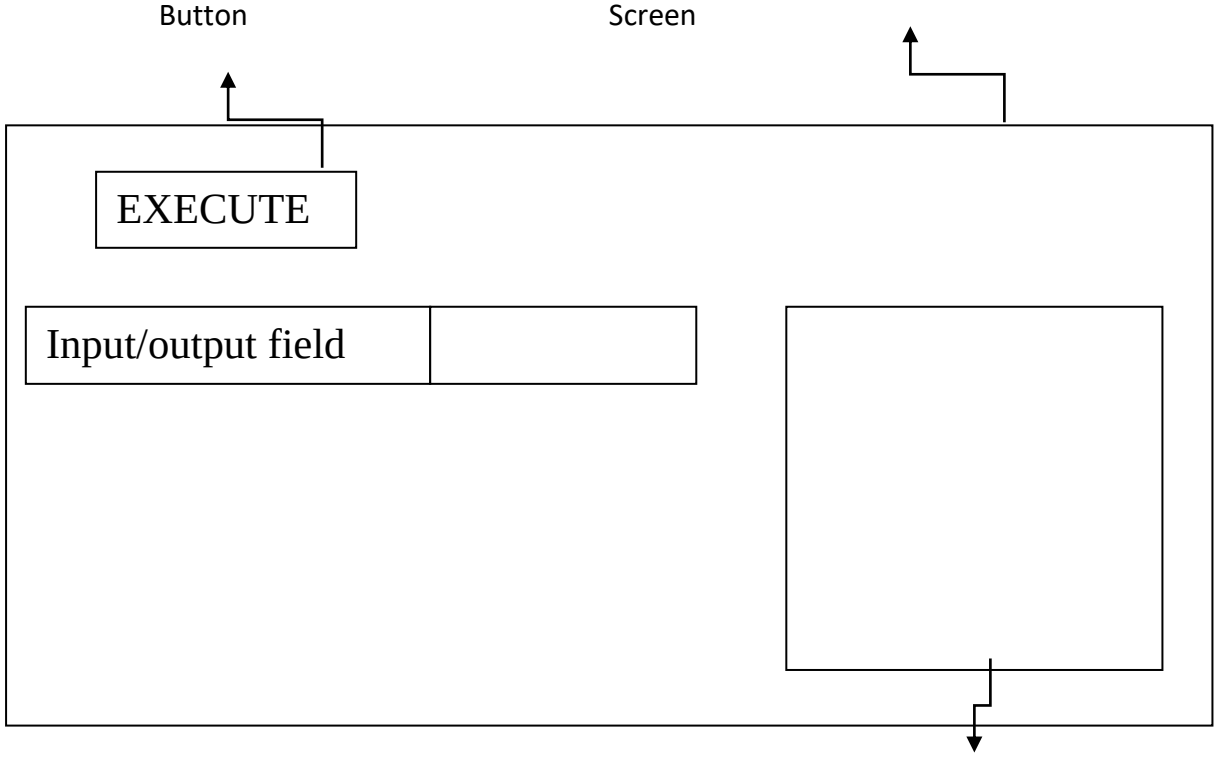
**2.LVC_S_LAYO**: This standard structure is used to populate the layout in OOALV.

**NOTE:** It can be compared with the type SLIS_LAYOUT_ALV of the type group SLIS in normal ALV.

**3.LVC_T_EVTS:** This table type is used to populate different events in OOALV.

**STEPS TO CREATE A CUSTOM CONTROL:-**

- Once the program is in change mode.
- Write the statement call screen followed by the screen number.
- Double click on the screen number.
- Click on yes.
- Press enter.
- Enter short description
- Click on Layout button.
- Click on custom control.
- Place it on the screen
- Expand the screen size as much as possible



- Enter the custom control name.
- Click on activate
- Press enter
- Click on flow logic.
- Specify and design the required PBO & PAO modules etc.,
- Activate the screen and click on back.
- The method 'free container' is used to refresh the container data

**NOTE** : The static method 'DOC_AT_LEFT' of the standard class

' CL_GUI_DOCKING_CONTAINER' is used to position the custom control at left hand side .

The static method 'DOC_ATRIGHT' of the standard class 'CL_GUI_DOCKING_CONTAINER' is used to position the custom control at left right side.

The reference object of a docking container must be passed to export parameters as a grid object

Ex:

*create a grid instance

CREATE OBJECT<rf_grid>

EXPORTNG

i_parent =rf_dock_container " CL_GUI_ALV_GRID

**NOTE**: the static method 'ORIENTATION_VERTICAL' is a standard class used to split the custom container in vertical.

The static method 'ORIENTATION_HORIZONTAL' is a standard class used to split the custom container in horizontal.

**SYNTAX:**

CREATE OBJECT<rf_split>

EXPORTING

PARENT = <rf_split>bottom_right_container

ORIENTATION = CL_GUI_EASY_SPLTTER_CONTAINER →
ORIENTATION_HORIZONTAL/ ORIENTATION_VERTICAL.

**NOTE**: the instance method 'TOP_LEFT_CONTAINER'
'BOTTOM_RIGHT_CONTAINER' as the splitter class
'CL_GUI_SPILTTER_CONTAINER' to be passed to an export parameter I_PARENT as the split class.

**SYNTAX:**

Gird at top splitter

CREATE OBJECT<rf_grid>

EXPORTING

I_parent =<rf_split> => 'TOP_LEFT_CONTAINER/BOTTOM_RIGHT_CONTAINER.

**CREATE DYNAMIC TABLE OF A CLASS CL_ALV_TABLE_CREATE:**

- This method is used to generate dynamic internal table based on the fields populated in a filed catalogue of the type "LVC_T_FCAT".

**STEPS TONPOPULATE THE DYNAMIC INTERNAL TABLE AND DATA**

- Declare (or) define filed catalogue of the type " LVC_T_FACT".
- Declare (or) define the field symbol line and standard table

Eg:

DATA:<gt_dyn_tab> TYPE LVC_T_FACT,

<gs_dyn_fcat> LIKE LINE OF<gt_dyn_fcat>.

*field symbols.

FILED-SYMBOLS: <<gs_line>>,

<<gs_dyn_table>> TYPE Standard table,

- Populte filed catalog with fixed and dynamic fileds by setting a loop.

Fixed coloumns.

<gs_dyn_fcat>-col_pos = '1".

<gs_dyn_fcat>-datatype/int type = '<char/....>'.

<gs_dyn_fcat>-itun = '< 10/20/30....>'.

.

.

APPEND <<gs_dyn_fcat> to <gt_dyn_fcat>.

*dynamic coulmns

LOOP AT <gt_data> into <gs_data>,

.

.

<gs_dyn_fact>-filedname= <gs_data-filed>.

APPEND <g_dyn_fcat> TO<gs_dyn_fcat>.

.

ENDLOOP.

- Use static method "CREATE _DYNAMIC_TABLE" as the .class " CL_ALV_TABLE_CREATE" to generate an internal table using the populated filed catalogue.

EG:

CALL METHOD C_ALV_TABKE_CRETE ===>crete _dynamic_table

EXPORTING

it_fieldcatalog = <gt_dyn_fcat>" dynamic filed catalog.

IMPORTING

cp_tabe = <it_table>"of the type ref to data.

ASSIGN <it_table> --><<gfs_dyn_table>>

CREATE data is line_n Like line of gfs_dyn_table

AASIGN is_line_n ---> to gfs_line.

- The data can be populated int dynamic internal table as below

Eg:

ASSIGHN Component <filedname.OF structure <<gfs_lines>> TO <<gfs_temp>>.

<<gfs_tmp>> = "profit".

APPEND <<gfs_line>> to<<gfs_gyn_table>>.

# Connect for more ..!!

**Swapnil More**
SAP ABAP Developer @SVKM
| S/4 Hana |OO-ABAP |


**Swapnil More**
SAP ABAP Developer @SVKM
| S/4 Hana |OO-ABAP |