Understanding the concepts of Object Oriented Programming

What is Object Orientation?
In the past, information systems used to be defined primarily by their functionality: Data and functions were kept separate and linked together by means of input and output relations.

The object-oriented approach, however, focuses on objects that represent abstract or concrete things of the real world. These objects are first defined by their character and their properties, which are represented by their internal structure and their attributes (data). The behavior of these objects is described by methods (functionality).

Comparison between Procedural and Object Oriented Programming

<table>
<thead>
<tr>
<th>Features</th>
<th>Procedure Oriented approach</th>
<th>Object Oriented approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emphasis</td>
<td>Emphasis on tasks</td>
<td>Emphasis on things that does those tasks.</td>
</tr>
<tr>
<td>Modularization</td>
<td>Programs are divided into smaller programs known as functions</td>
<td>Programs are organized into classes and objects and the functionalities are embedded into methods of a class.</td>
</tr>
<tr>
<td>Data security</td>
<td>Most of the functions share global data</td>
<td>Data can be hidden and cannot be accessed by external sources.</td>
</tr>
<tr>
<td>Extensibility</td>
<td>Relatively more time consuming to modify for extending existing functionality.</td>
<td>New data and functions can be easily added whenever necessary</td>
</tr>
</tbody>
</table>

Object Oriented Approach - key features

2. Real world entity can be modeled very well.
3. Stress on data security and access.
4. Reduction in code redundancy.
5. Data encapsulation and abstraction.

What are Objects and Classes?

**Objects:** An object is a section of source code that contains data and provides services. The data forms the attributes of the object. The services are known as methods (also known as operations or functions). They form a capsule which
combines the character to the respective behavior. Objects should enable programmers to map a real problem and its proposed software solution on a one-to-one basis.

**Classes**: Classes describe objects. From a technical point of view, objects are runtime instances of a class. In theory, you can create any number of objects based on a single class. Each instance (object) of a class has a unique identity and its own set of values for its attributes.

### Local and Global Classes

As mentioned earlier a class is an abstract description of an object. Classes in ABAP Objects can be declared either globally or locally.

**Global Class**: Global classes and interfaces are defined in the Class Builder (Transaction SE24) in the ABAP Workbench. They are stored centrally in class pools in the class library in the R/3 Repository. All of the ABAP programs in an R/3 System can access the global classes.

**Local Class**: Local classes are define in an ABAP program (Transaction SE38) and can only be used in the program in which they are defined.

<table>
<thead>
<tr>
<th></th>
<th>Global Class</th>
<th>Local Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessed By</td>
<td>Any program</td>
<td>Only the program where it is defined.</td>
</tr>
<tr>
<td>Stored In</td>
<td>In the Class Repository</td>
<td>Only in the program where it is defined.</td>
</tr>
<tr>
<td>Created By</td>
<td>Created using transaction SE24</td>
<td>Created using SE38</td>
</tr>
<tr>
<td>Namespace</td>
<td>Must begin with Y or Z</td>
<td>Can begin with any character</td>
</tr>
</tbody>
</table>

### Local Classes

Every class will have two sections.

1. **Definition**: This section is used to declare the components of the classes such as attributes, methods, events. They are enclosed in the ABAP statements CLASS ... ENDC.

   ```abap
   CLASS <class> DEFINITION.
   ...
   ENDC.
   ...
   ENDCLASS.
   ```

2. **Implementation**: This section of a class contains the implementation of all methods of the class. The implementation part of a local class is a processing block.

   ```abap
   CLASS <class> IMPLEMENTATION.
   ...
   ENDC.
   ```
Structure of a Class

The following statements define the structure of a class:

1. A class contains components
2. Each component is assigned to a visibility section
3. Classes implement methods

1. Components of a Class are as follow:

□ Attributes:- Any data, constants, types declared within a class form the attribute of the class.

□ Methods:- Block of code, providing some functionality offered by the class. Can be compared to function modules. They can access all of the attributes of a class.

   Methods are defined in the definition part of a class and implement it in the implementation part using the following processing block:

   METHOD <meth>.

   ...

   ENDMETHOD.

   Methods are called using the CALL METHOD statement.

□ Events:- A mechanism set within a class which can help a class to trigger methods of other class.

□ Interfaces:- Interfaces are independent structures that you can implement in a class to extend the scope of that class.

   Instance and Static Components:

□ Instance components exist separately in each instance (object) of the class and are referred using instance component selector using ‘□’.

□ Static components only exist once per class and are valid for all instances of the class. They are declared with the CLASS- keywords

□ Static components can be used without even creating an instance of the class and are referred to using static component selector ‘⇒’.

2. Visibility of Components

Each class component has a visibility. In ABAP Objects the whole class definition is separated into three visibility sections: PUBLIC, PROTECTED, and PRIVATE.
Data declared in public section can be accessed by the class itself, by its subclasses as well as by other users outside the class.

Data declared in the protected section can be accessed by the class itself, and also by its subclasses but not by external users outside the class.

Data declared in the private section can be accessed by the class only, but not by its subclasses and by external users outside the class.

CLASS <class> DEFINITION.
PUBLIC SECTION.
...
PROTECTED SECTION.
...
PRIVATE SECTION.
...
ENDCLASS.

We shall see an example on **Visibility of Components** once we become familiar with attributes of ABAP Objects.
The yellow block of code is CLASS Definition

The Green block of code is CLASS Implementation

The Grey block of code is for object creation. This object creation includes two steps:

Step 1 is Create a reference variable with reference to the class.

**Syntax:** DATA: <object name> TYPE REF TO <class name>.

Step 2: Create an object from the reference variable:

**Syntax:** CREATE OBJECT <object name>.

Output for the above code is

<table>
<thead>
<tr>
<th>Demo for local class creation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABAP Objects</td>
</tr>
<tr>
<td>This is method &quot;DISPLAY&quot;</td>
</tr>
</tbody>
</table>

**Attributes of Object Oriented Programming:**

- Inheritance.
- Abstraction.
- Encapsulation.
- Polymorphism

**Inheritance** is the concept of adopting the features from the parent and reusing them. It involves passing the behavior of a class to another class. You can use an existing class to derive a new class. Derived classes inherit the data and methods of the super class. However, they can overwrite existing methods, and also add new ones.

Inheritance is of two types: Single Inheritance and Multiple Inheritance
Single Inheriting: Acquiring the properties from a single parent. (Children can be more).

Example for Single Inheritance

**Multiple inheritance**: Acquiring the properties from more than one parent.

Example

Tomato4 (Best Color, Size, Taste)

Tomato1
(Best color)

Tomato2
(Best Size)

Tomato3
(Best Taste)
Syntax: CLASS <subclass> DEFINITION INHERITING FROM <superclass>.

Let us see a very simple example for creating subclass(child) from a superclass(parent)
Multiple Inheritance is **not supported** by ABAP.

Output is as follows:
Abstraction: Everything is visualized in terms of classes and objects.

Encapsulation: The wrapping up of data and methods into a single unit (called class) is known as Encapsulation. The data is not accessible to the outside world only those methods, which are wrapped in the class, can access it.

Polymorphism: Methods of same name behave differently in different classes. Identical (identically-named) methods behave differently in different classes. Object-oriented programming contains constructions called interfaces. They enable you to address methods with the same name in different objects. Although the form of address is always the same, the implementation of the method is specific to a particular class.
Object oriented programming (OOP) explained with an example

Create a class that keeps track of a bank account balance. Then write a program to use this class.

Steps involved:

- Run the class builder utility (**SE24**).
- Create a class called ZACCOUNTxx, where xx is the last two digits of your logon ID.
- Declare a PRIVATE attribute BALANCE of type DMBTR to store the account balance.
- Create the following PUBLIC methods:
  - SET_BALANCE (Sets the balance to a new value)
    - IMPORTING NEW_BALANCE TYPE DMBTR
  - DEPOSIT (Adds a deposit amount to the balance and returns the new balance)
    - IMPORTING AMOUNT TYPE DMBTR
    - RETURNING NEW_BALANCE TYPE DMBTR
  - WITHDRAW (Subtracts a deposit amount from the balance and returns the new balance.)
    - IMPORTING AMOUNT TYPE DMBTR
    - RETURNING NEW_BALANCE TYPE DMBTR
    - EXCEPTIONS INSUFFICIENT_FUNDS
- Activate all elements of your class.
- Write a program called Z_USE_ACCOUNT_xx, where xx is the last two digits of your logon ID. This program should do the following:
  - Instantiate an instance of the Account class.
  - Set the account balance to some initial value.
  - Make several deposits and withdrawals, printing the new balance each time. Do not allow the balance to become less than zero. (Use the exception to detect this.)
- Test and debug your program.

"Extra Credit": If you have extra time, try any of the following:

- Replace the SET_BALANCE method with a constructor. Pass the opening balance when you instantiate the account object.
- Create a static attribute and methods to set and get the name of the bank that holds the accounts.

Step-by-step approach with screen-shots

Go to SE24 (Class builder)

Type in ZACCOUNTAA as the name of the class and press Create.
Define 3 methods DEPOSIT, SET_BALANCE and WITHDRAW.

Place the mouse cursor in DEPOSIT and hit the Parameters button.

Write the parameters imported / exported for DEPOSIT method.

Similarly for SET_BALANCE

And WITHDRAW
For withdraw we define an exception.

We can see the attributes and methods by pressing “Display object list” button on top.

Now we IMPLEMENT the 3 methods. Double click the method DEPOSIT.

```
method DEPOSIT
  BALANCE = BALANCE + AMOUNT.
  NEW_BALANCE = BALANCE.
endmethod.
```
Write the required code. Similarly for SET_BALANCE

```plaintext
method SET_BALANCE
    BALANCE = NEW_BALANCE.
endmethod.
```

Similarly for WITHDRAW.

```plaintext
method WITHDRAW
    IF BALANCE >= AMOUNT.
    BALANCE = BALANCE - AMOUNT.
    NEW_BALANCE = BALANCE.
    ELSE.
    RAISE INSUFFICIENT_FUNDS.
    ENDIF.
endmethod.
```

Now we are almost done creating the object. Press CTRL + F3 to activate or hit the Matchstick.

We will see this in the status `Active object generated`

Now we are done building the global class we can test it. Press F8.
**Test Class ZACCOUNTAA**

Click SET_BALANCE. Write the NEW_BALANCE and press ENTER.

We come back to Initial Screen. Now click DEPOSIT.

We see the return Values now.

Now let's WITHDRAW 4000
Now the BALANCE is 2000

Let’s try withdrawing 3000 now.

We get an exception.

Given below is an example code for using the global class we defined.

```
REPORT ZGB_OOPS_BANK .

DATA: acct1 type ref to zaccountaa.

DATA: bal type i.

create object: acct1.
```
selection-screen begin of block a.
parameters: p_amnt type dmbtr,
          p_dpst type dmbtr,
          p_wdrw type dmbtr.
selection-screen end of block a.

start-of-selection.

  call method acct1->set_balance( p_amnt ).
  write:/ 'Set balance to ', p_amnt.

  bal = acct1->deposit( p_dpst ).
  write:/ 'Deposited ', p_dpst ,' bucks making balance to ', bal.

  bal = acct1->withdraw( p_wdrw ).
  write:/ 'Withdrew ', p_wdrw ,' bucks making balance to ', bal.

This is the output.

Program ZGB_OOPS_BANK

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Set balance to</td>
<td>4,000.00</td>
</tr>
<tr>
<td>Deposited</td>
<td>3,000.00</td>
</tr>
<tr>
<td>Withdraw</td>
<td>200.00</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Demo program illustrating Simple class and Super class

*&----------------------------------------------------------&*
*& Report  Z_OOABAP18  &
*& &
*&----------------------------------------------------------&*
*&
*&----------------------------------------------------------&*
*& REPORT  Z_OOABAP18 .
CLASS lcl_employee DEFINITION.
PUBLIC SECTION.

* The public section is accessible from outside

*------------------------------------------------------

TYPES:
  BEGIN OF t_employee,
    no  TYPE i,
    name TYPE string,
  END OF t_employee.
METHODS:
  constructor
    IMPORTING im_employee_no TYPE i
    im_employee_name TYPE string,
    display_employee.
* Class methods are global for all instances
CLASS-METHODS: display_no_of_employees.

PROTECTED SECTION.

* The protected section is accessible from the class and its subclasses

*------------------------------------------------------

* Class data are global for all instances
CLASS-DATA: g_no_of_employees TYPE i.
PRIVATE SECTION.

* The private section is only accessible from within the class

*------------------------------------------------------

DATA: g_employee TYPE t_employee.
ENDCLASS.

--- LCL Employee - Implementation
CLASS lcl_employee IMPLEMENTATION.
METHOD constructor.
  g_employee-no = im_employee_no.
  g_employee-name = im_employee_name.
  g_no_of_employees = g_no_of_employees + 1.
ENDMETHOD.
METHOD display_employee.
  WRITE:/ 'Employee', g_employee-no, g_employee-name.
ENDMETHOD.
METHOD display_no_of_employees.
WRITE: / 'Number of employees is:', g_no_of_employees.
ENDMETHOD.
ENDCLASS.

************************************
************************************
* R E P O R T
************************************
************************************
DATA: g_employee1 TYPE REF TO lcl_employee,
      g_employee2 TYPE REF TO lcl_employee.
START-OF-SELECTION.
CREATE OBJECT g_employee1
  EXPORTING im_employee_no = 1
      im_employee_name = 'Vikram.C'.
CREATE OBJECT g_employee2
  EXPORTING im_employee_no = 2
      im_employee_name = 'Raghava.V'.
CALL METHOD g_employee1->display_employee.
CALL METHOD g_employee2->display_employee.
REPORT Z_OOABAP19.
CLASS lcl_company_employees DEFINITION.
PUBLIC SECTION.
TYPES:
  BEGIN OF t_employee,
    no TYPE i,
    name TYPE string,
    wage TYPE i,
  END OF t_employee.
METHODS:
  constructor,
  add_employee IMPORTING im_no TYPE i
                 im_name TYPE string
                 im_wage TYPE i,
  display_employee_list,
  display_no_of_employees.
PRIVATE SECTION.
  CLASS-DATA: i_employee_list TYPE TABLE OF t_employee,
              no_of_employees TYPE i.
ENDCLASS.
*-- CLASS LCL_CompanyEmployees IMPLEMENTATION
CLASS lcl_company_employees IMPLEMENTATION.
METHOD constructor.
  no_of_employees = no_of_employees + 1.
ENDMETHOD.
METHOD add_employee.
  Adds a new employee to the list of employees
DATA: l_employee TYPE t_employee.
  l_employee-no = im_no.
  l_employee-name = im_name.
  l_employee-wage = im_wage.
  APPEND l_employee TO i_employee_list.
ENDMETHOD.
METHOD display_employee_list.
  Displays all employees and there wage
DATA: l_employee TYPE t_employee.
WRITE: / 'List of Employees'.
LOOP AT i_employee_list INTO l_employee.
  WRITE: / l_employee-no, l_employee-name, l_employee-wage.
ENDLOOP.
METHOD display_no_of_employees.
* Displays total number of employees
  SKIP 3.
  WRITE: / 'Total number of employees:', no_of_employees.
ENDMETHOD.
ENDCLASS.

*******************************************************
* Sub class LCL_BlueCollar_Employee
*******************************************************
CLASS lcl_bluecollar_employee DEFINITION
  INHERITING FROM lcl_company_employees.
PUBLIC SECTION.
METHODS:
  constructor
    IMPORTING im_no TYPE i
    im_name TYPE string
    im_hours TYPE i
    im_hourly_payment TYPE i,
    add_employee REDEFINITION.
PRIVATE SECTION.
DATA: no TYPE i,
    name TYPE string,
    hours TYPE i,
    hourly_payment TYPE i.
ENDCLASS.

*---- CLASS LCL_BlueCollar_Employee IMPLEMENTATION
CLASS lcl_bluecollar_employee IMPLEMENTATION.
  METHOD constructor.
  * The superclass constructor method must be called from the subclass
    CALL METHOD super->constructor.
    no = im_no.
    name = im_name.
    hours = im_hours.
    hourly_payment = im_hourly_payment.
ENDMETHOD.
  METHOD add_employee.
  * Calculate wage an call the superclass method add_employee to add
    the employee to the employee list
    DATA: l_wage TYPE i.
    l_wage = hours * hourly_payment.
    CALL METHOD super->add_employee
      EXPORTING im_no = no
      im_name = name
      im_wage = l_wage.
* Sub class LCL_WhiteCollar_Employee

CLASS lcl_whitecollar_employee DEFINITION

  INHERITING FROM lcl_company_employees.

  PUBLIC SECTION.

  METHODS:

  constructor
  IMPORTING
    im_no TYPE i
    im_name TYPE string
    im_monthly_salary TYPE i
    im_monthly_deducations TYPE i,
  add_employee REDEFINITION.

  PRIVATE SECTION.

  DATA:

    no TYPE i,
    name TYPE string,
    monthly_salary TYPE i,
    monthly_deducations TYPE i.

ENDCLASS.

*---- CLASS LCL_WhiteCollar_Employee IMPLEMENTATION

METHOD constructor.

* The superclass constructor method must be called from the subclass
* constructor method

  CALL METHOD super->constructor.

  no = im_no.
  name = im_name.
  monthly_salary = im_monthly_salary.
  monthly_deducations = im_monthly_deducations.

ENDMETHOD.

METHOD add_employee.

* Calculate wage an call the superclass method add_employee to add

  the employee to the employee list

  DATA: _wage TYPE i.

  _wage = monthly_salary - monthly_deducations.

  CALL METHOD super->add_employee

  EXPORTING
    im_no = no
    im_name = name
    im_wage = _wage.

ENDMETHOD.

ENDCLASS.

REDIT

*******************************************************

* R E P O R T

*******************************************************

DATA:
* Object references
  o_bluecollar_employee1 TYPE REF TO lcl_bluecollar_employee,
o_whitecollar_employee1 TYPE REF TO lcl_whitecollar_employee.
START-OF-SELECTION.
* Create bluecollar employee object
CREATE OBJECT o_bluecollar_employee1
  EXPORTING im_no = 1
  im_name = 'Vikram.C'
  im_hours = 38
  im_hourly_payment = 75.
* Add bluecollar employee to employee list
CALL METHOD o_bluecollar_employee1-&gt;add_employee
  EXPORTING im_no = 1
  im_name = 'Vikram.C'
  im_wage = 0.
* Create whitecollar employee object
CREATE OBJECT o_whitecollar_employee1
  EXPORTING im_no = 2
  im_name = 'Raghava.V'
  im_monthly_salary = 10000
  im_monthly_deducations = 2500.
* Add bluecollar employee to employee list
CALL METHOD o_whitecollar_employee1-&gt;add_employee
  EXPORTING im_no = 1
  im_name = 'Vikram.C'
  im_wage = 0.
* Display employee list and number of employees. Note that the result
  will be the same when called from o_whitecollar_employee1 or
  o_bluecollar_employee1, because the methods are defined
  as static (CLASS-METHODS)
CALL METHOD o_whitecollar_employee1-&gt;display_employee_list.
CALL METHOD o_whitecollar_employee1-&gt;display_no_of_employees.
Demo program illustrating Interface

REPORT Z_OOABAP20.

INTERFACE lif_employee.
  METHODS:
    add_employee
      IMPORTING im_no  TYPE i
      im_name  TYPE string
      im_wage  TYPE i.
  ENDINTERFACE.

*******************************************************
* Super class LCL_CompanyEmployees
*******************************************************

CLASS lcl_company_employees DEFINITION.
  PUBLIC SECTION.
    INTERFACES lif_employee.
    TYPES:
      BEGIN OF t_employee,
        no  TYPE i,
        name TYPE string,
        wage TYPE i,
      END OF t_employee.
    METHODS:
      constructor,
      display_employee_list,
      display_no_of_employees.
  PRIVATE SECTION.
    CLASS-DATA: i_employee_list TYPE TABLE OF t_employee,
                no_of_employees TYPE i.
  ENDCLASS.

*-- CLASS LCL_CompanyEmployees IMPLEMENTATION

CLASS lcl_company_employees IMPLEMENTATION.
  METHOD constructor.
    no_of_employees = no_of_employees + 1.
  ENDMETHOD.
  METHOD lif_employee~add_employee.
* Adds a new employee to the list of employees
  DATA: l_employee TYPE t_employee.
  l_employee-no = im_no.
  l_employee-name = im_name.
  l_employee-wage = im_wage.
  APPEND l_employee TO i_employee_list.
ENDMETHOD.

METHOD display_employee_list.
* Displays all employees and their wage
  DATA: l_employee TYPE t_employee.
  WRITE: / 'List of Employees'.
  LOOP AT i_employee_list INTO l_employee.
    WRITE: / l_employee-no, l_employee-name, l_employee-wage.
  ENDLOOP.
ENDMETHOD.

METHOD display_no_of_employees.
* Displays total number of employees
  SKIP 3.
  WRITE: / 'Total number of employees:', no_of_employees.
ENDMETHOD.

ENDCLASS.

*******************************************************
* Sub class LCL_BlueCollar_Employee
*******************************************************

CLASS lcl_bluecollar_employee DEFINITION
  INHERITING FROM lcl_company_employees.
PUBLIC SECTION.
METHODS:
  constructor
    IMPORTING im_no TYPE i
    im_name TYPE string
    im_hours TYPE i
    im_hourly_payment TYPE i,
    lif_employee~add_employee REDEFINITION.
PRIVATE SECTION.
  DATA:no TYPE i,
  name TYPE string,
  hours TYPE i,
  hourly_payment TYPE i.
ENDCLASS.

*---- CLASS LCL_BlueCollar_Employee IMPLEMENTATION
CLASS lcl_bluecollar_employee IMPLEMENTATION.
METHOD constructor.
* The superclass constructor method must be called from the subclass
* constructor method
  CALL METHOD super->constructor.
  no = im_no.
  name = im_name.
hours = im_hours.
hourly_payment = im_hourly_payment.
ENDMETHOD.
METHOD lif_employee~add_employee.
* Calculate wage an call the superclass method add_employee to add
* the employee to the employee list
DATA: l_wage TYPE i.
l_wage = hours * hourly_payment.
CALL METHOD super->lif_employee~add_employee
  EXPORTING im_no = no
      im_name = name
      im_wage = l_wage.
ENDMETHOD.
ENDCLASS.

*******************************************************************************
* Sub class LCL_WhiteCollar_Employee
*******************************************************************************
CLASS lcl_whitecollar_employee DEFINITION
  INHERITING FROM lcl_company_employees.
PUBLIC SECTION.
METHODS:
  constructor
    IMPORTING im_no TYPE i
        im_name TYPE string
        im_monthly_salary TYPE i
        im_monthly_deducations TYPE i,
    lif_employee~add_employee REDEFINITION.
PRIVATE SECTION.
DATA:
  no TYPE i,
  name TYPE string,
  monthly_salary TYPE i,
  monthly_deducations TYPE i.
ENDCLASS.
*---- CLASS LCL_WhiteCollar_Employee IMPLEMENTATION
CLASS lcl_whitecollar_employee IMPLEMENTATION.
METHOD constructor.
* The superclass constructor method must be called from the subclass
* constructor method

   CALL METHOD super->constructor.
   no = im_no.
   name = im_name.
   monthly_salary = im_monthly_salary.
   monthly_deducations = im_monthly_deducations.
ENDMETHOD.
METHOD lif_employee~add_employee.
* Calculate wage an call the superclass method add_employee to add
* the employee to the employee list
DATA: l_wage TYPE i.
l_wage = monthly_salary - monthly_deducations.
CALL METHOD super->lif_employee~add_employee
  EXPORTING im_no = no
  im_name = name
  im_wage = l_wage.
ENDMETHOD.
ENDCCLASS.

******************************************************
* R E P O R T
******************************************************
DATA:
* Object references
  o_bluecollar_employee1 TYPE REF TO lcl_bluecollar_employee,
  o_whitecollar_employee1 TYPE REF TO lcl_whitecollar_employee.
START-OF-SELECTION.
* Create bluecollar employee object
  CREATE OBJECT o_bluecollar_employee1
    EXPORTING im_no = 1
      im_name = 'Chandrasekhar'
      im_hours = 38
      im_hourly_payment = 75.
* Add bluecollar employee to employee list
  CALL METHOD o_bluecollar_employee1->lif_employee~add_employee
    EXPORTING im_no = 1
      im_name = 'Vikram C'
      im_wage = 0.
* Create whitecollar employee object
  CREATE OBJECT o_whitecollar_employee1
    EXPORTING im_no = 2
      im_name = 'Raghava V'
      im_monthly_salary = 10000
      im_monthly_deducations = 2500.
* Add whitecollar employee to employee list
  CALL METHOD o_whitecollar_employee1->lif_employee~add_employee
    EXPORTING im_no = 1
      im_name = 'Gylle Karen'
      im_wage = 0.
* Display employee list and number of employees. Note that the result
  will be the same when called from o_whitecollar_employee1 or
  o_bluecollar_employee1, because the methods are defined
  as static (CLASS-METHODS)
  CALL METHOD o_whitecollar_employee1->display_employee_list.
  CALL METHOD o_whitecollar_employee1->display_no_of_employees.
Global Class Functionality (Step-by-step approach)

Go to **SE24** T-Code.

Provide the name.

Click on the **Create** button.

Provide the Description.
Press **Save** button. Then we can view the screen like this.

Provide method.

Goto **Attributes**

Provide the values.
In ZCL_KNA1 is the structure.

And ZCL_TT is table type.

Go to methods tab.
And double click on the method select method.

And write the logic.

The code is like this.

Go back

Save check and activate it.

And provide another method Display method.

Double click on the display method.

Then write the logic.
Save it, check it, activate it.

Provide the logic in se38.

Create the program.
Provide the logic.

Then save it, check it, activate it.

And execute it.

The output is like this.
Provide the values.

Execute it.
Working with the Keyword **SUPER** in object Oriented Programming

**SUPER** is the key word used to represent the super class of a class in oops you can access the methods and attributes of the super class using this word **SUPER**.

Press CREATE.

Save it.
Provide parameter for this method.

Double click on the method then provide the logic.

Save it, check it. And Activate it.

Go to SE24.

Provide another sub class.

In this we can provide super class name in the sub class attributes.
Save it.

Then we can see the methods tab.

In this we can get automatically the super class method.

Go to attributes tab.

Then provide the variables.

Save it.
Go to the methods.

Provide the logic in the method double click.

Save it, check it and activate it.

Here we can use SUPER keyword.

Then go to SE38.

Provide the logic in this program.

```
*&----------------------------------------------------------------------*
*& Report  ZCL_SUB_METHOD                                           *
*&                                                                       *
*&----------------------------------------------------------------------*
*&   How to work with SUPER keyword                                  *
*&                                                                       *
*&----------------------------------------------------------------------*
REPORT  ZCL_SUB_METHOD .
*Provide object for sub class
DATA: OBJ TYPE REF TO ZCL_SUB_METHOD.
*provide parameters
PARAMETERS: P_VBELN TYPE VBAK-VBELN.
*Provide data object
DATA: WA_VBAK TYPE VBAK,
     WA_VBAP TYPE VBAP,
     IT_VBAP TYPE Z_VBAP.
```
*Create the object
CREATE OBJECT OBJ.
*Call select method
CALL METHOD OBJ->SELECT_METHOD
  EXPORTING
    P_VBELN = P_VBELN
  IMPORTING
    WA_VBAK = WA_VBAK.
*Display header data
WRITE:/ WA_VBAK-VBELN,
      WA_VBAK-ERDAT,
      WA_VBAK-ERZET,
      WA_VBAK-ERNAM.
SKIP 2.
*Provide item data
IT_VBAP = OBJ->IT_VBAP."For Your Reference this IT_VBAP is declared in attribute
*Display item data
LOOP AT IT_VBAP INTO WA_VBAP.
WRITE:/ WA_VBAP-VBELN,
      WA_VBAP-POSNR,
      WA_VBAP-MATKL.
ENDLOOP.

Then save it, check it, and activate it.
Here one important point is by using one object in the sub class.
Then we can implement the super class method automatically.
The output for this program is as follows.

Provide the values.

Execute it.
this is the program for how 'SUPER' keyword work

SAPTechnical

this is the program for how 'SUPER' keyword work

4970  03.01.1997 12:52:54 CURA

4970  000010
4970  000020
4970  000030
4970  000040
**Working with Inheritance**

**Inheritance** is the concept of passing the behavior of a class to another class.

- You can use an existing class to derive a new class.
- Derived class inherits the data and methods of a super class.
- However, they can overwrite the methods existing methods and also add new ones.
- Inheritance is to inherit the attributes and methods from a parent class.

Inheritance:

- Inheritance is the process by which objects of one class acquire the properties of another class.
- Advantage of this property is **reusability**.
- This means we can add additional features to an existing class without modifying it.

Go to SE38.

Provide the program name.

Provide the properties.

Save it.

Provide the logic for inheritance.

```plaintext
REPORT ZLOCALCLASS_VARIABLES.
OOPS INHERITANCE
*SUPER CLASS FUNCTIONALITY
*DEFINE THE CLASS.
CLASS CL_LC DEFINITION.
PUBLIC SECTION.
DATA: A TYPE I,
     B TYPE I,
     C TYPE I.
METHODS: DISPLAY,
     MM1.
CLASS-METHODS: MM2.
ENDCLASS.
*CLASS IMPLEMENTATION
CLASS CL_LC IMPLEMENTATION.
METHOD DISPLAY.
WRITE:/ 'THIS IS SUPER CLASS' COLOR 7.
ENDMETHOD.
METHOD MM1.
```
WRITE: / 'THIS IS MM1 METHOD IN SUPER CLASS'.
ENDMETHOD.
METHOD MM2.
WRITE: / 'THIS IS THE STATIC METHOD' COLOR 2.
WRITE: / 'THIS IS MM2 METHOD IN SUPER CLASS' COLOR 2.
ENDMETHOD.
ENDCLASS.
*SUB CLASS FUNCTIONALITY
*CREATE THE CLASS.
*INHERITING THE SUPER CLASS.
CLASS CL_SUB DEFINITION INHERITING FROM CL_LC. "HOW WE CAN INHERIT
PUBLIC SECTION.
DATA: A1 TYPE I,
    B1 TYPE I,
    C1 TYPE I.
METHODS: DISPLAY REDEFINITION, "REDEFINE THE SUPER CLASS METHOD
SUB.
ENDCLASS.
*CLASS IMPLEMENTATION.
CLASS CL_SUB IMPLEMENTATION.
METHOD DISPLAY.
WRITE: / 'THIS IS THE SUB CLASS OVERWRITE METHOD' COLOR 3.
ENDMETHOD.
METHOD SUB.
WRITE: / 'THIS IS THE SUB CLASS METHOD' COLOR 3.
ENDMETHOD.
ENDCLASS.
*CREATE THE OBJECT FOR SUB CLASS.
DATA: OBJ TYPE REF TO CL_SUB.
START-OF-SELECTION.
CREATE OBJECT OBJ.
CALL METHOD OBJ->DISPLAY. "THIS IS SUB CLASS METHOD
CALL METHOD OBJ->SUB.
WRITE: / 'THIS IS THE SUPER CLASS METHODS CALLED BY THE SUB CLASS OBJECT' COLOR 5.
SKIP 1.
CALL METHOD OBJ->MM1. "THIS IS SUPER CLASS METHOD
CALL METHOD OBJ->MM2.
*CREATE THE OBJECT FOR SUPER CLASS.
DATA: OBJ1 TYPE REF TO CL_LC.
START-OF-SELECTION.
CREATE OBJECT OBJ1.
SKIP 3.
WRITE: / 'WE CAN CALL ONLY SUPER CLASS METHODS BY USING SUPER CLASS OBJECT' COLOR 5.
CALL METHOD OBJ1->DISPLAY. "THIS IS SUPER CLASS METHOD
CALL METHOD OBJ1->MM1.
CALL METHOD OBJ1->MM2.

Save it, check it, activate it and execute it.

Then the output is like this.
simple local class program using variables

THIS IS THE SUB CLASS OVERRIDE METHOD
THIS IS THE SUB CLASS METHOD
THIS IS THE SUPER CLASS METHODS CALLED BY THE SUB CLASS OBJECT

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

WE CAN CALL ONLY SUPER CLASS METHODS BY USING SUPER CLASS OBJECT
THIS IS SUPER CLASS
THIS IS MM1 METHOD IN SUPER CLASS
THIS IS THE STATIC METHOD
THIS IS MM2 METHOD IN SUPER CLASS
Working with constructor

Description of Constructor:

- Constructor is automatically called when an object created.
- Constructor is the same name of the class.
- No return value.
- With in static method we can only access class attributes.
- Class-constructor does not have any parameters.
- Constructor has only import parameters.

Go to SE38 provide program name and property.

Save it.

Provide the logic.

```abap
*&---------------------------------------------------------------------*&
*& Report  ZLOCALCLASS_VARIABLES* &
*&
*&---------------------------------------------------------------------*&
*& How to work Constructor* &
*VikramChellappa* &
*&---------------------------------------------------------------------* &
REPORT ZLOCALCLASS_VARIABLES.
OOPS CONSTRUCTOR.
**PROVIDE DATA TYPES  "CONSTRUCTOR DOES NOT HAVE ANY EXPORT PARAMETERS.
*DATA: C TYPE I.
*DEFINE THE CLASS.
CLASS CL_LC DEFINITION.
PUBLIC SECTION.
METHODS: CONSTRUCTOR IMPORTING A TYPE I,
* EXPORTING B TYPE I, "IT TAKES ONLY IMPORT PARAMETERS
ANOTHER.
ENDCLASS.
*class implementation.
CLASS CL_LC IMPLEMENTATION.
METHOD CONSTRUCTOR.
WRITE:/ 'THIS IS CONSTRUCTOR METHOD'.
WRITE:/ 'A =', A.
ENDMETHOD.
METHOD ANOTHER.
WRITE:/ 'THIS IS ANOTHER METHOD' COLOR 5.
ENDMETHOD.
ENDCLASS.
*create the object.
DATA OBJ TYPE REF TO CL_LC.
START-OF-SELECTION.
CREATE OBJECT OBJ EXPORTING A = 10.
*  IMPORTING B = C.
*call the method.
```
SKIP 2.
CALL METHOD OBJ->ANOTHER.

Save it, check it, activate it.

Execute it.

Then the output is like this.
Insert data into the database table using Classes

Go to Class Builder and create a new class.

Provide the method name.

Go to parameters and provide the attributes.

Go back to methods. And provide the logic by double click on the method name.
Then save it, check it, activate it and execute it.

Press F8.

The data is stored in database.
To verify, go to VBAK table (SE16) and check whether the data is stored or not.

Now we will create a program using the above class for inserting the data into the database table.

Go to SE38 and create a program.

Select create button.

After that provide the following logic.

```abap
*&------------------------------------------------------------------*
*& Report ZPG_INSERTINTODB *
*&
*&------------------------------------------------------------------*
*&
*&------------------------------------------------------------------*
REPORT ZPG_INSERTINTODB.
*provide the object for the class
DATA: OBJ_INSERT TYPE REF TO ZCL_INSERTDB.
*provide parameters
PARAMETERS: V_VBELN TYPE VBELN,
            V_ERDAT TYPE ERDAT,
            V_ERZET TYPE ERZET.
*provide work area
DATA: WA TYPE VBAK.
```
*create the object
START-OF-SELECTION.
CREATE OBJECT OBJ_INSERT.
*provide insert method
CALL METHOD OBJ_INSERT->INSERT_DATA
*provide exporting parameters
EXPORTING
  P_VBELN = V_VBELN
  P_ERDAT = V_ERDAT
  P_ERZET = V_ERZET
*provide import parameters
IMPORTING
  WA_VBAK = WA.
*display the data.
WRITE:/ WA-VBELN,
      WA-ERDAT,
      WA-ERZET.

Save it, activate it, execute it.

The screen is like this.

Provide values.

Execute it.
Following is the sample output of the same:
Go to SE38 and create a program.

Then provide the following code.

REPORT  ZLOCALCLASS_VARIABLES.
*How we can use import and export and changing parameters in the class.
*Provide the variables
DATA: V_IMP TYPE I,
    V_CHA TYPE I VALUE 100.
*Define the class.
CLASS CL_LC DEFINITION.
PUBLIC SECTION.
METHODS: DISPLAY IMPORTING A TYPE I
    EXPORTING B TYPE I
    CHANGING C TYPE I.
ENDCLASS.
*Implement the class.
CLASS CL_LC IMPLEMENTATION.
METHOD DISPLAY.
  B = A + 20.
  C = A + 30.
ENDMETHOD.
ENDCLASS.
*Create the object.
DATA OBJ TYPE REF TO CL_LC.
START-OF-SELECTION.
CREATE OBJECT OBJ.
CALL METHOD OBJ->DISPLAY
EXPORTING
  A = 10
IMPORTING
  B = V_IMP
CHANGING
  C = V_CHA.
WRITE:/ 'OUTPUT PARAMETR', V_IMP,
       'CHANGING PARAMETER', V_CHA.

Save and activate the program.

Now execute the program by inserting a breakpoint.
Press F5.
Press F5
Press F5.
The values are changed.

Press F5.

Then
Final output.
Working on Polymorphism

POLYMORPHISM:-

Polymorphism is a characteristic of being able to assign a different behavior or value in a subclass, to something that was declared in a parent class. For example, a method can be declared in a parent class, but each subclass can have a different implementation of that method. This allows each subclass to differ, without the parent class being explicitly aware that a difference exists.

CLAUSES REGARDING POLYMORPHISM:-

1. Allows one interface to be used for a general class of actions.
2. When objects from different classes react differently to the same procedural call.
3. User can work with different classes in a similar way, regardless of their implementation.
4. Allows improved code organization and readability as well as creation of “extensible” programs.
5. Although the form of address is always the same, the implementation of the method is specific to a particular class.

Go to SE24 T-code.

Press create button.
Press **Save** button.

Provide methods.

Select the first method then provide the parameters for this method.

Go back to the methods then double click on the method name.

Then provide the logic.
Select **display method** then provide the parameters for this method.

Go back to method then provide the logic.

Save it ☐, check it ☐, and activate it ☑.

Provide **SUBCLASS**:

Press **CREATE** button.
Click on ![SAVE](image)

Go to attribute provide the values that means provide super class name.

Go to methods we can see like this.
Select select _ method then press the REDEFINE button.

Then screen would like this.

Provide the logic.
Then save it.

Go to Attributes.

Then provide the Variables.

Go back to the methods.

Then provide another method.
Double click on the method then provide the logic

Click on SAVE, CHECK, and ACTIVATE.

Then provide the code in the T-Code SE38.

Provide the logic.

*Provide Object for Sub Class

DATA: OBJ1 TYPE REF TO ZCL_POLYMORPHISM_SUB.

*Provide Parameters
PARAMETERS: V_VBELN TYPE VBAP-VBELN.
*Provide Data Objects
DATA: WA_VBAP TYPE VBAP,
   IT_VBAP TYPE Z_VBAP.
*Create the Object
CREATE OBJECT OBJ1.
*Call the Redefine Select Method
CALL METHOD OBJ1->SELECT_METHOD
  EXPORTING
   P_VBELN = V_VBELN
  * IMPORTING
  * WA_VBAK =.
*Provide the IT_VBAP Values
IT_VBAP = OBJ1->IT_VBAP.
LOOP AT IT_VBAP INTO WA_VBAP.
  WRITE:/ WA_VBAP-VBELN,
    WA_VBAP-POSNR,
    WA_VBAP-MATNR.
ENDLOOP.

Click On **SAVE**, **CHECK**, **ACTIVATE** and **EXECUTE** it.

Output :-

The output data display in the list.
Enhancement of a Standard Class

Go to TCode SE24.

Enter the name of the Class to be enhanced.

The Following Screen would be displayed.

Click on Class > Enhance.

Give the name of the enhancement implementation and short text

Click on continue.
Enter the New method “GET_DATA_NEW”

Click on Parameters.
Enter the Required parameters, Type, Associated Type.
Click on Back and get back to method screen.
Enter the Source code.

Click on Save Check and activate.

Create a Report program on SE38 T- Code.

Click on Pattern.

Select ABAP Object Patterns.
Click on continue.

Enter the Enhance Class name and method name.

Click on continue.
PASTE THE BELOW CODE.

*---------------------------------------------------------------------*
* Report  ZENHANCE_TEST
* DEMO FOR ENHANCING THE STANDARD CLASS.
REPORT  ZENHANCE_TEST.
* TYPE DECLARATIONS
DATA : TABLE TYPE STRING,
  ROW_COUNT TYPE I,
  DATA_OUT TYPE TABLE OF SFLIGHT,
  W_OUT LIKE LINE OF DATA_OUT.
* Calling the Enhanced class and Enhanced methods.
CALL METHOD CL_WDR_FLIGHTS=>GET_DATA_NEW
  EXPORTING
*   ROW_COUNT =
   TAB_NAME  = 'SFLIGHT'
  CHANGING
   DATA      = DATA_OUT.
LOOP AT DATA_OUT INTO W_OUT.
WRITE :/ W_OUT-CARRID, W_OUT-FLDATE.
ENDLOOP.

Click on Save Check and Activate.

Execute the program:
ABAP Classes in Workflow

1. **ABAP Classes and Business Workflow:**

We can use **ABAP classes** in the definition and runtime components of **SAP Web Flow Engine** in the same way as object types defined in the Business object Repository (BOR).

Before proceeding further we need to know where to create and maintain **ABAP Classes** and **ABAP Interfaces**.

2. **What is Class Builder and its purpose?**

The **Class Builder** allows us to create and maintain global ABAP classes and interfaces. Both of these object types, like global data types, are defined in the **ABAP Repository**, thus composing a central class library. Together, they form a central class library and are visible throughout the system. We can display existing classes and interfaces in the class library using the Class Browser.

We can define local classes as well as global classes. They are defined locally in programs, function groups or as auxiliary classes of global classes of the class pools. Local classes are only visible within the defining module.

ABAP classes are processed using the Class Builder.

3. **How to reach Class Builder?**

To reach the initial screen of the Class Builder, choose Development □ Class Builder from the initial screen of the ABAP Workbench or enter transaction code **SE24**.
4. **How does it integrate?**

The Class Builder allows us to create Web development objects within the ABAP Workbench. We can use the Class Browser to display and maintain existing global object types from the class library.

The diagram below illustrates the architecture of the Class Builder and the relationships between its components (including the Class Browser).

![Diagram of Class Builder architecture](image)

From here, we can either display the contents of the class library or edit a class using the Class Editor. Once we have defined an object type, we can implement its methods. From the initial screen or the Class Editor, we can also access the Class Builder’s test environment. We can define the object types immediately after implementing the method in the ABAP Editor. It is also possible to access the test environment from the initial screen or Class Editor.

5. **How to use the Class Builder?**

Use the Class Builder to:

- Display an overview (in the Class Browser) of global object types and their relationships.
• Maintain existing global classes or interfaces.
• Create new global classes and interfaces.
• Implement inheritance between global classes.
• Create compound interfaces.
• Create and specify the attributes, methods, and events of global classes and interfaces.
• Define internal types in classes.
• Implement methods.
• Redefine methods.
• Maintain local auxiliary classes.
• Test classes or interfaces in a simulated runtime environment.

6. **What are the constraints?**

We cannot define object types on the basis of graphical object modeling.

7. **Note before creating global classes and interfaces:**

Global classes and interfaces that we create in the Class Builder are stored in the class library and administered by the R/3 Repository: they therefore have the same namespace as all other Repository objects. It is therefore necessary to have naming conventions for object types and their components and to use them uniformly within program development.

8. **Naming Conventions in ABAP Objects:**

The following naming convention has been conceived for use within the **SAP namespace**. If we do not observe the naming conventions for object types (classes and interfaces), conflicts will occur when the system creates persistent classes, since it will be unable to generate the necessary co-classes.

9. **Conventions for Object Types:**

<table>
<thead>
<tr>
<th>Class in the class library</th>
<th>CL_&lt;class name&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces in the class library</td>
<td>IF_&lt;interface name&gt;</td>
</tr>
<tr>
<td>Local classes in programs (recommendation)</td>
<td>LCL_&lt;class name&gt;</td>
</tr>
<tr>
<td>Local interfaces in programs (recommendation)</td>
<td>LIF_&lt;interface name&gt;</td>
</tr>
</tbody>
</table>
10. **Conventions for Components:**

<table>
<thead>
<tr>
<th>Method name</th>
<th>&lt;method name&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>&lt;event name&gt;</td>
</tr>
<tr>
<td>Local type definitions within a class (recommendation)</td>
<td>TY_&lt;type name&gt;</td>
</tr>
<tr>
<td>Data definitions (variables)</td>
<td>&lt;variable name&gt;</td>
</tr>
<tr>
<td>Data definitions (constants) (recommendation)</td>
<td>CO_&lt;constant name&gt;</td>
</tr>
</tbody>
</table>

11. **Local Conventions within Methods:**

For parameters:

| IMPORTING parameters | IM_<parameter name> |
| EXPORTING parameters | EX_<parameter name> |
| CHANGING parameters  | CH_<parameter name> |
| RESULT               | RE_<result> |

12. **Using ABAP Classes in Workflow:**

Within the SAP WebFlow Engine we can use ABAP classes that support the IF_WORKFLOW interface. Classes that have implemented the IF_WORKFLOW interface are recognized as workflow-enabled in the Class Builder.

13. **How to create ABAP Classes that support IF_WORKFLOW interface?**
   - Go to transaction **SE24** and create a customized class.
Next the pop up appears where we need to mention the detail as follows:

- Save it and the class is created.

- Now the class is to implement IF_WORKFLOW interface. For this go to the Interfaces tab and declare the IF_WORKFLOW as the interface there and press Enter; two sub-interfaces appear: BI_OBJECT and BI_PERSISTENT. Save the Class.
The ZCL_TEST class now contains the existing methods of IF_WORKFLOW interface.

14. **Lights on Key Attributes and Attributes:**

   The key attributes are used to define the object key. There can also be other defined attributes other than key attributes. The SAP Web Flow Engine can access all public attributes of a class.

   **Key Attributes:**

   In the Class Builder there is an additional column **Key Attributes** on the **Attributes** tab page as shown below:
We need to **check** this box when we are defining any attribute as the Key Attribute.

All key fields must be character-type fields (elementary types: CHAR, NUMC) and have a defined length. The maximum length allowed for all key fields is 32 characters. The length of the key field for the persistent display is 32 characters.

In the case of persistent ABAP objects we can use the GUID, which is generated automatically by the object manager when an instance is created.

**Attributes:**

In addition to all the other data types that the Class Builder supports, we can also define attributes with reference to an object from the Business Object Repository (BOR). To do this, we have to use the structure SWOTOBJID as the data type. The BOR object is determined using the corresponding value.

To assign a BOR object instance to an attribute we need to use the corresponding BOR macros. Normally, this is implemented within the CONSTRUCTOR of a class.

To use the BOR macros in a class, two INCLUDES must be included.

- □ Include <CNTN03>................contains the local types
- □ Include <CNTN02>................contains the BOR macros
An example to show how to define Attributes and Key Attributes:

15. Why IF_WORKFLOW Interface?

The IF_WORKFLOW interface is necessary when using an ABAP class within the SAP Web Flow Engine. The interface contains methods that allow the object to be used within the SAP Web Flow Engine.

The SAP Web Flow Engine handles all objects generically. Objects have to be saved in the event of a context change. Therefore, it is necessary to convert object references in such a way that they can be saved persistently. Conversely, we have to be able to generate the corresponding instance of an ABAP class from the persistently saved key.

There are also a number of SAP Web Flow Engine components, for example, the Workflow Log that can display objects. In this case the object has to provide corresponding functions.

The IF_WORKFLOW interface puts a logical parenthesis round the BI_PERSISTENT (instance management) and BI_OBJECT (object behavior) interfaces. The IF_WORKFLOW interface contains the following methods:

- `BI_PERSISTENT~FIND_BY_LPOR`
- `BI_PERSISTENT~LPOR`
- `BI_PERSISTENT~REFRESH`
- `BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE`
A class that implements the IF_WORKFLOW interface can be used in any workflow. The class is automatically released for use in workflows when the interface is implemented. Therefore, we can only make compatible changes to a class after implementation (we cannot delete attributes, change types or delete methods). There is no where-used list to show which workflows the class is used in.

Internal classes of an application should not implement the IF_WORKFLOW interface, since this could mean that each method of the class is used in the workflow. Therefore, we should encapsulate the workflow functions in another class that calls the selected methods of the internal class.

Each method of the IF_WORKFLOW Interface as mentioned earlier has its distinct functionality, which is discussed below.

16. **BI_PERSISTENT~FIND_BY_LPOR Method:**

If we want to convert a persistent saved display of an object into an instance of the corresponding ABAP class, SAP Web flow Engine calls the BI_PERSISTENT~FIND_BY_LPOR method.

**Features:**

The method parameter LPOR is the persistent object reference and is of SIBFLPOR structure type. A reference of BI_PERSISTENT type is returned.

The following table shows the components of the SIBFLPOR structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATID</td>
<td>Describes the object type (CL for ABAP classes)</td>
</tr>
<tr>
<td>TYPEID</td>
<td>ABAP class name</td>
</tr>
<tr>
<td>INSTID</td>
<td>Object key. The key is limited to 32 characters.</td>
</tr>
</tbody>
</table>

We can implement this method in several ways. In the case of persistent classes we can create the ABAP object instance using the generated classes. In the case of individual persistence management we have to implement the individual actions (such as creating an instance, performing an existence check, entering public attributes, and so on) manually within the class.

Instance management takes place automatically in the case of persistent classes. In the case of individual persistence management we also have to carry out instance management by class. The SAP Web Flow Engine does not provide any instance management. We must therefore implement our own instance management in the case of individual persistence management.
The `FIND_BY_LPOR` method should always return the same instance if the following problems are to be avoided:

- Inconsistency in the data display
- Instance data being overwritten by another instance
- Locking conflicts

There is an implementation example in the `CL_SWF_FORMABSENC` demo class.

Class Builder: Class CL_SWF_FORMABSENC Display

```plaintext
METHOD bi_persistent-find_by_lpor.
  "- begin of local data
  DATA: 1_formabsenc TYPE REF TO cl_swf_formabsenc.
  DATA: 1_number TYPE swxformabs-formnumber.
  DATA: 1_instance TYPE t_instance.
  "- end of local data

  CHECK 1por-instid IS NOT INITIAL.

  " instid is the key of the object
  1_number = 1por-instid.

  READ TABLE mst_instances WITH KEY number = 1_number INTO 1_instance.
  IF sy-subrc <> 0.
    TRY.
      CREATE OBJECT 1_formabsenc
      EXPORTING
      1m_number = 1_number
    CATCH cx_bo_error.
    "------- object not found
    EXIT.
    ENDTry.

  1_instance-number = 1_number.
  1_instance-instance = 1_formabsenc.
  APPEND 1_instance TO mst_instances.
ENDIF.

  result = 1_instance-instance.
ENDMETHOD.
```
17. **BI_PERSISTENT~LPOR Method:**

The BI_PERSISTENT~LPOR method is the counterpart to the BI_PERSISTENT~FIND_BY_LPOR method. It provides the persistent display for an existing instance of an ABAP object.

**Features:**

The method returns the persistent display of an object reference as a SIBFLPOR type structure as described earlier.

There is a close relationship between the BI_PERSISTENT~FIND_BY_LPOR method and the BI_PERSISTENT~LPOR method. If we call the BI_PERSISTENT~FIND_BY_LPOR method first and then the BI_PERSISTENT~LPOR method, the BI_PERSISTENT~LPOR method must return the same value as was previously used to call the BI_PERSISTENT~FIND_BY_LPOR method.

There are also several ways of implementing this method in this case. There is an implementation example in the CL_SWF_FORMABSENC demo class.

### Class Builder: Class CL_SWF_FORMABSENC Display

```
METHOD bi_persistent~lpor .
   result = me->m_por.
ENDMETHOD.
```

18. **BI_PERSISTENT~REFRESH Method:**

SAP Web Flow Engine calls the BI_PERSISTENT~REFRESH method when the system has to ensure that all values of an object are valid or that they agree exactly with the persistent display of the object.

**Features:**

The method implementation depends on the internal organization of the class. We can check the object instance data in the database, if necessary.
If we do not need the method in our class, then we need only to carry out a “dummy” implementation (without further coding) to avoid program errors when the system calls the method.

There is an implementation example in the CL_SWF_FORMABSENCE demo class.

19. BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE Method:

The BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE method returns the display name of the object.

**Features:**

We can display references to process objects or process step objects at different positions within the SAP Web Flow Engine (for example, in Business Workplace and in Workflow Log). The object key is normally displayed here. If, for example, we want to display a descriptive text instead, the BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE method has to return the corresponding value.

If the method does not contain implementation or does not return a value, the object key is displayed.

If we do not need the method in our class, then we need only to carry out a “dummy” implementation (without further coding) to avoid program errors when the system calls the method.

There is an implementation example in the CL_SWF_FORMABSENCE demo class.
20. **BI_OBJECT~EXECUTE_DEFAULT_METHOD Method:**

The **BI_OBJECT~EXECUTE_DEFAULT_METHOD** method is the standard method for the object. This method is executed when, for example, we call the object in Business Workplace.

**Features:**

We can display process objects or process step objects at different positions within the SAP Web Flow Engine (for example, in Business Workplace and in Workflow Log). The SAP Web Flow Engine calls the **BI_OBJECT~EXECUTE_DEFAULT_METHOD** method.

If we do not need the method in our class, then we need only to carry out a “dummy” implementation (without further coding) to avoid program errors when the system calls the method.

There is an implementation example in the **CL_SWF_FORMABSENCE** demo class.
21. **BI_OBJECT~RELEASE Method:**

The system indicates that the reference to the instance is no longer needed by using the **BI_OBJECT~RELEASE** method. This means we can delete the reference from instance management. Once the last reference has been deleted from instance management, the **GARBAGE COLLECTOR** can release the corresponding memory area.

**Features:**

If we do not need the method in our class, then we need only to carry out a “dummy” implementation (without further coding) to avoid program errors when the system calls the method.

There is an implementation example in the **CL_SWF_FORMABSENC** demo class.
22. **How to use ABAP Classes in Process Steps of Business Workflow?**

In process steps we can use methods and attributes of ABAP classes in the same way as methods and attributes of Business Object Repository (BOR) objects. We can call these methods in the process context.

**Features:**

While using the ABAP Classes in the Process Steps the methods may contain dialogs, they can be synchronous or asynchronous; they may appear in the workflow log, and so on.

In general, we can use any method that is implemented as a public method. The method can be implemented in the class itself, in one of the super classes of the class, or by way of an interface.

The maximum permitted length for methods that are implemented by way of an interface, for example `IF_WORKFLOW~FIND_BY_LPOR`, is 30 characters. If the method name is too long, we can choose a shorter name for the method by defining an alias. If the method is implemented in the class or in a super class, the name of the method cannot be longer than 30 characters, so this limitation does not apply.

**Parameters:**

We can assign values from the workflow container to the method parameters. Conversely, export parameters can be entered as workflow container values. The following overview shows how the individual types can be used as parameters:

- Simple types (string, integer, and so on)
- Data Dictionary types (structures, tables, complex types)
- References to objects from the Business Object Repository
- References to ABAP classes (supporting the IF_WORKFLOW interface)

We can transfer method parameters that represent a persistent object (IF_WORKFLOW or BOR Object) in the following ways:

- ABAP classes (with IF_WORKFLOW interface)
  - TYPE SIBFLPORB
    - Object is transferred using the persistent display
  - TYPE REF TO <Class name>
Object is transferred as object reference

- **BOR objects**
  - **TYPE SIBFLPORB**
    - Object is transferred using the persistent display
  - **TYPE SWOTOBJID**
    - Object is transferred using the persistent display; this display is only valid for BOR objects
  - **TYPE SWC_OBJECT**
    - Object is transferred as object reference

**Exceptions:**

The SAP Web Flow Engine can deal with exceptions that are triggered by the methods. It differentiates between application exceptions and temporary exceptions. The two exception categories are differentiated by the exception in the class hierarchy or by naming conventions. In the case of a temporary exception, the SAP Web Flow Engine attempts to execute the method again. In the case of a permanent error the status for the workflow is set to error.

**Class-Based Exceptions:**

To create a temporary exception, we can use, for example, the `CX_BO_TEMPORARY` class or a corresponding subclass. It can be helpful to trigger an exception for dialog methods when the user cancels the dialog. Here, for example, we could trigger the `CX_BO_ACTION_CANCELED` exception (subclass of the `CX_BO_TEMPORARY` class).

**Exceptions Not Based on Class:**

We can also trigger exceptions not based on class. The SAP Web Flow Engine can differentiate between the two exception categories (temporary and permanent) by the name. If the exception begins with TMP or TEMP, it is a temporary exception; otherwise it is a permanent exception.
“I would like to explain about Working with Events in Global Class”.

Go to Class Builder “SE24”.

Provide class name.

Press create button.

Save it.

Go to event tab.

Then provide event method.
Provide parameters also for this method.

Save it.

Then go to methods option.

We wouldn’t be able to write any code in the events directly.

For this we can create another method in the method tab.

Then provide link between method and also the event method.

Then we can click on this detail view button.
Then enable the event handler for check box.

Provide the class name and also the event name.
Save & activate. Following screen appears:

Now select the method.
And also copy the parameters of the event method.
By pressing this copy event parameter we can get the parameters.

Save and go back to the earlier screen.

Then double click on the method name.

Then provide the following logic for triggering the event.

METHOD METHOD_EVENT.

*check the condition

IF S_LIFNR_LOW < 1000 AND S_LIFNR_HIGH > 2000.

MESSAGE I000(0) WITH 'enter the values between 1000 and 2000'.

RAISE EVENT ZEVENT_METHOD.

ENDIF.

*provide select statement

SELECT *

FROM LFA1
INTO TABLE IT_LFA1
WHERE LIFNR BETWEEN S_LIFNR_LOW AND S_LIFNR_HIGH.

*transfer the values to another internal table
IT_LFA11 = IT_LFA1.
ENDMETHOD.

After that provide the logic in se38.

REPORT  ZCL_EVENT_OPERATION  .

*provide data objects
DATA: LFA1 TYPE LFA1,
  OBJ TYPE REF TO ZCL_EVENT_OPERATION,
  IT_LFA1 TYPE Z_LFA1,
  IT_LFA11 TYPE Z_LFA1,
  WA_LFA1 TYPE LFA1.

*provide select statement
SELECT-OPTIONS: S_LIFNR FOR LFA1-LIFNR.

*provide create object
START-OF-SELECTION.
  CREATE OBJECT OBJ.

*call the method
CALL METHOD OBJ->METHOD_EVENT
  EXPORTING
    S_LIFNR_LOW = S_LIFNR-LOW
    S_LIFNR_HIGH = S_LIFNR-HIGH
    IT_LFA11 = IT_LFA1.

*provide attribute value
IT_LFA11 = OBJ->IT_LFA11.

*display the data

LOOP AT IT_LFA11 INTO WA_LFA1.

WRITE:/ WA_LFA1-LIFNR,
       WA_LFA1-LAND1,
       WA_LFA1-NAME1,
       WA_LFA1-ORT01.

ENDLOOP.

Save it, check it, activate it and execute it.

Then the output is like this.

If lifnr value is <1000 and >2000.
Then press execute it.

The output is like this.

Then press enter.
Working with Interfaces

In ABAP interfaces are implemented in addition to, and independently of classes. An interface only has a declaration part, and do not have visibility sections. Components (Attributes, methods, constants, types) can be defined the same way as in classes.

- Interfaces are listed in the definition part of the class, and must always be in the PUBLIC SECTION.
- Operations defined in the interface are implemented as methods of the class. All methods of the interface must be present in the implementation part of the class.
- Attributes, events, constants and types defined in the interface are automatically available to the class carrying out the implementation.
- Interface components are addressed in the class by <interface name>~<component name>

Go to SE24 provide interface name.

Interface name start with ZIF_

Press create button.

Provide description.

Save it.
Save it.

Provide the method name.

Provide the parameters for this method.
The screen is like this.

Then save it, check it, activate it.

We cannot implement the method of this interface.

Provide the name in the class.

Create it.
Save it.

Go to interface tab.

Provide the interface name.

Save it.

Then go to the methods tab.

Then we can see the interface method name in the class method.

Then double click on the method then write the logic here.
Then save it, check it, activate it.

Create a program in SE38.
Provide the code.

*&------------------------------------------------------------------*
*& Report  ZCL_INTERFACE                                     *
*&------------------------------------------------------------------*
 报告 ZCL_INTERFACE.
*provide mara table
DATA: MARA TYPE MARA.
*provide data objects
DATA: OBJ TYPE REF TO ZCL_INTERFACE,
   IT_MARA TYPE Z_MARA,
   WA_MARA TYPE MARA.
*provide selection screen
SELECT-OPTIONS: S_MATNR FOR MARA-MATNR.
*provide object
START-OF-SELECTION.
   CREATE OBJECT OBJ.
*call the method.
   CALL METHOD OBJ->ZIF_INTERFACE~SELECT_METHOD
      EXPORTING
         P_MATNR_LOW  = S_MATNR-LOW
         P_MATNR_HIGH = S_MATNR-HIGH
      IMPORTING
         IT_MARA      = IT_MARA
         WA_MARA      = WA_MARA.
*display the data
   LOOP AT IT_MARA INTO WA_MARA.
   WRITE:/ WA_MARA-MATNR,
            WA_MARA-ERSDA,
            WA_MARA-ERNAM,
            WA_MARA-MATKL,
            WA_MARA-MEINS.
   ENDLOOP.

Then save it, check it, activate it then execute it the output is like this.

The output is see in the list.
What is the use of aliases?

ALIASES:

This is the aliases name. It is only for interfaces.

Go to SE24.

Then go to aliases tab.

Then provide another name for the interface method.

Then provide public.

Save it, check it, activate it.

Then go to SE38.

Change the method name also.

```sql
*&---------------------------------------------------------------*
*& Report  ZCL_INTERFACE                                        *
*&                                                         *
*&---------------------------------------------------------------*
REPORT  ZCL_INTERFACE .
*provide mara table
DATA: MARA TYPE MARA.
*provide data objects
DATA: OBJ TYPE REF TO ZCL_INTERFACE,
```
IT_MARA TYPE Z_MARA,
WA_MARA TYPE MARA.
*provide selection screen
SELECT-OPTIONS: S_MATNR FOR MARA-MATNR.
*provide object
START-OF-SELECTION.
CREATE OBJECT OBJ.
*call the method.
* CALL METHOD OBJ->ZIF_INTERFACE~SELECT_METHOD
CALL METHOD OBJ->SEL
EXPORTING
  P_MATNR_LOW  = S_MATNR-LOW
  P_MATNR_HIGH = S_MATNR-HIGH
IMPORTING
  IT_MARA      = IT_MARA
  WA_MARA      = WA_MARA.
*display the data
LOOP AT IT_MARA INTO WA_MARA.
  WRITE:/ WA_MARA-MATNR,
         WA_MARA-ERSDA,
         WA_MARA-ERNAM,
         WA_MARA-MATKL,
         WA_MARA-MEINS.
ENDLOOP.

The output would be as shown below:

```
this is interface program in se24

  100-100 07.11.1994 BALLER 801 PC
  100-101 06.03.1996 KUNSTZ 801 PC
  100-110 07.11.1994 BALLER 801 PC
  100-120 08.11.1994 BALLER 801 PC
  100-130 08.11.1994 BALLER 801 PC
```
Creating a global class from a local class

In this tutorial, we would look into the procedure of creating a global class using a local class defined in a program.

Consider the following Z program, which contains a local class:

```
REPORT  zclass_test.
*---------------------------------------------------------*
*       CLASS zcl_test DEFINITION
*---------------------------------------------------------*
*---------------------------------------------------------*
CLASS zcl_test DEFINITION.
  PUBLIC SECTION.
  METHODS: display.
ENDCLASS.                    "zcl_test DEFINITION
*--------------------------------------------------------*
*       CLASS zcl_test IMPLEMENTATION
*--------------------------------------------------------*
*--------------------------------------------------------*
CLASS zcl_test IMPLEMENTATION.
  METHOD display.
    WRITE: 'SAPTechnical.com'.
ENDMETHOD.                    "display
ENDCLASS.                    "zcl_test IMPLEMENTATION
```

Now let us create a global class SE24 using the above local class:

Go to transaction SE24.

Now select Object type □ import □ Local classes in program (As shown below):
Following pop-up appears:

Enter your Z program in which the local class is defined and press ENTER.
The class name defined in our program is ZCL_TEST and the proposed global class name is CL_ZCL_TEST. Now you can rename the global class name as per your requirement.

If the local class is defined inside an include, we need to check the checkbox “Explode INCLUDEs”.

Now click on import. Following message would appear:

Import completed successfully

Now check the global class ZCL_TEST.
Create Transaction for local class method

In this demo I am going to show how to create transaction on a local class method.

**Step1:** First create a local class in a report from transaction SE38.
REPORT z_demo_oop_jg.

```
**---------------------------------------------------------------------**
**       CLASS create_report DEFINITION**
**---------------------------------------------------------------------**
**---------------------------------------------------------------------**
CLASS create_report DEFINITION.
PUBLIC SECTION.
  METHODS: main.
PRIVATE SECTION.
  DATA: i_data TYPE STANDARD TABLE OF SBOOK INITIAL SIZE 0.
  METHODS: fetch_data,
            display_data.
ENDCLASS.                   "create_report DEFINITION
**---------------------------------------------------------------------**
**       CLASS create_report IMPLEMENTATION**
**---------------------------------------------------------------------**
**---------------------------------------------------------------------**
CLASS create_report IMPLEMENTATION.
  METHOD fetch_data.
* Select 100 records from SBOOK table
  SELECT * FROM SBOOK
  INTO TABLE i_data
  UP TO 100 ROWS.
ENDMETHOD.                   "fetch_data
  METHOD display_data.
   CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
      EXPORTING
                  i_structure_name = 'SBOOK'
      TABLES
                  t_outtab = i_data
      EXCEPTIONS
                  program_error = 1
                  OTHERS = 2.
      IF sy-subrc <> 0.
       MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
       WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
      ENDIF.
ENDMETHOD.                   "display_data
  METHOD main.
   fetch_data( ).
   display_data( ).
ENDMETHOD.                   "main
ENDCLASS.                   "create_report IMPLEMENTATION
Step2. Now from transaction SE93 create a transaction for the method MAIN as shown in the screen shots given below:

Give a transaction name and press create button.

In the next screen give a description and choose the proper radio button.

In the next screen provide report name (where the local class is defined), local class name and method name.
transaction and execute it.
In this case it will display the report.

This technique can be used to call a method (local class) from another program using
statement: call transaction.

EX: call transaction 'Z_OOP'.

Note: In the same way you can create a transaction on method of a global class.
**Persistent Objects: A Quick Reference**

**Objective**
To store references to the persistent object persistently in the database.

**Step: 1 -> Create a database table**

This table should contain 2 fields of type OS_GUID in addition to the GUID object attribute. The first field is used to store the instance GUID while the other is used to store the class GUID.

**Step: 2 -> Create a Persistent Class**

In the next screen select the class type as Persistent Class and then hit Save Button.
Step: 3 -> Persistent Mapping or Mapping

Goto->Persistence Representation

Give the table name. For e.g. ZSTUDENT03 and hit the enter button
Table fields appear in the lower half of the tool screen. Double Click the table field and press the button. Add the remaining fields.

While adding the field INST_GUID choose the assignment type as Object reference and for the attribute type specify the class name for e.g. ZCL_PERSIST_03
To assign a class indicator, select the corresponding table field of type **OS_GUID** by double-clicking. Enter the name of the reference attribute for the attribute name.

Screen looks like below. Press Save.

Activate the Class. Press the Yes Button to activate the class actor as well.

Write a Program to create the persistent object.
Source Code excerpt:

DATA: AGENT TYPE REF TO ZCA_PERSIST_03,
    STUDENT TYPE REF TO ZCL_PERSIST_03,
    REF1 TYPE REF TO OBJECT.
DATA: SNO LIKE ZSTUDENT04-SNO VALUE '1000',
    SNAME LIKE ZSTUDENT04-SNAME VALUE 'HAKIM',
    MARK1 LIKE ZSTUDENT04-MARK1 VALUE '100',
    MARK2 LIKE ZSTUDENT04-MARK2 VALUE '100'.
AGENT = ZCA_PERSIST_03=>AGENT.
TRY.
CALL METHOD AGENT->CREATE_PERSISTENT
    EXPORTING
*   I_INST_GUID =
    I_MARK1     = MARK1
    I_MARK2     = MARK2
    I_SNAME     = SNAME
    I_SNO       = SNO
*  RECEIVING
*    RESULT      =
    COMMIT WORK.
CATCH CX_OS_OBJECT_EXISTING .
ENDTRY.

Go to SE16 and check the entries.

Table ZSTUDENT03 Display

<table>
<thead>
<tr>
<th>Client</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUID</td>
<td>(GUID)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student Number</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student Name</td>
<td>HAKIM</td>
</tr>
<tr>
<td>Mark (MARK1)</td>
<td>100</td>
</tr>
<tr>
<td>Mark (MARK2)</td>
<td>100</td>
</tr>
</tbody>
</table>

| GUID | (INST GUID) | 00000000000000000000000000000000 |
| GUID | (CLASS GUID) | 00000000000000000000000000000000 |

Store the Persistent Object Reference in the database.

Source Code excerpt.

TRY.
CALL METHOD AGENT->IF_OS_CA_PERSISTENCY~GET_PERSISTENT_BY_OID
    EXPORTING
I_OID = '30EA9E25999F0843BE6F7B86063F2916'
RECEIVING
RESULT = REF1
CATCH CX_OS_OBJECT_NOT_FOUND.
CATCH CX_OS_CLASS_NOT_FOUND.
ENDTRY.
STUDENT ?= REF1.
STUDENT->SET_INST_GUID(STUDENT).
COMMIT WORK.

Go to SE16 and check the entries.

Table ZSTUDENT03 Display

<table>
<thead>
<tr>
<th>Client</th>
<th>GUID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30EA9E25999F0843BE6F7B86063F2916</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student Number</th>
<th>GUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>30EA9E25999F0843BE6F7B86063F2916</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student Name</th>
<th>MARK1</th>
<th>MARK2</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAKIM</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GUID</th>
<th>INST GUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>30EA9E25999F0843BE6F7B86063F2916</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GUID</th>
<th>CLASS GUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDAC954CFF36DA479FF4BD78D022C24E7</td>
<td></td>
</tr>
</tbody>
</table>
Persistent Objects: Using Business Key Identity

Objective

To Store the attributes of the Objects persistently in the database.

Step 1 -> Create a Persistent Class

Go to Class Builder (TCode SE24)

Give persistent class name for e.g. ZCL_PERSIST_01 and hit the create button

In the next screen select the class type as Persistent Class and then hit Save Button.
Step: 2 -> Persistent Mapping or Mapping

Utilities->Persistence Representation

Give the table name. For e.g. ZSTUDENT01 and hit the enter button

Table fields appear below the mapping screen.
Double Click the table field and then press the upward arrow button ▲

Add the remaining fields as well. Screen looks like this now.

Activate the Class. Press the Yes Button to activate the class actor as well.
Step: 3 -> Write a Program to create / fetch / delete the Persistent Object

Our Program Selection-Screen looks like below:

Here I am creating a new student. Specify the value and hit the execute button.

Output:

Persistent Objects

<table>
<thead>
<tr>
<th>Selection-Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student Number</td>
</tr>
<tr>
<td>Student Name</td>
</tr>
<tr>
<td>Mark1</td>
</tr>
<tr>
<td>Mark2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Selection-Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch Persistent</td>
</tr>
<tr>
<td>Create Persistent</td>
</tr>
<tr>
<td>Delete Persistent</td>
</tr>
</tbody>
</table>

Object Created
Go to SE16 and check the entries

Data Browser: Table ZSTUDENT01 Select Entries

<table>
<thead>
<tr>
<th>Student Number</th>
<th>Student Name</th>
<th>Mark1</th>
<th>Mark2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1090</td>
<td>ABDUL</td>
<td>99</td>
<td>99</td>
</tr>
</tbody>
</table>

Source Code

*---------------------------------------------------------------*
*& Report  Z_GET_PERSISTENT
*& Published @ SAPTechnical.com
*---------------------------------------------------------------*
*&Author : Abdul Hakim
*&Development Language: ABAP
*&System Release: SAP Netweaver 2004
*&Title: Persistent Object using Business Key Object Identity!!
*---------------------------------------------------------------*

REPORT  Z_GET_PERSISTENT.
selection-screen begin of block blk1 with frame title tit1.
parameters: sno   like zstudent01-sno obligatory,
   sname like zstudent01-sname obligatory,
   mark1 like zstudent01-mark1 obligatory,
   mark2 like zstudent01-mark2 obligatory.
selection-screen end of block blk1.
selection-screen begin of block blk2 with frame title tit2.
parameters: r1 type c radiobutton group rad1,
   r2 type c radiobutton group rad1,
   r3 type c radiobutton group rad1.
selection-screen end of block blk2.
*---------------------------------------------------------------*
*   CLASS lcl_class1 DEFINITION
*---------------------------------------------------------------*
*---------------------------------------------------------------*
class lcl_class1 definition.
public section.
data: agent     type ref to zca_persist_01,
   students  type ref to zcl_persist_01.
data result1 type ref to zcl_persist_01.
methods: fetch_persistent importing im_sno like sno
   im_sname like sname,
create_persistent importing im_sno like sno
   im_sname like sname
   im_mark1 like mark1
   im_mark2 like mark2,
delete_persistent importing im_sno like sno
   im_sname like sname,

output.
private section.
data: sno type zstudent01-sno,
   sname type zstudent01-sname,
   mark1 type zstudent01-mark1,
   mark2 type zstudent01-mark2.
endclass.                    "lcl_class1 DEFINITION
*---------------------------------------------------------------------*
*       CLASS lcl_class1 IMPLEMENTATION                                      
*---------------------------------------------------------------------*
*
*---------------------------------------------------------------------*
class lcl_class1 implementation.
method fetch_persistent.
   agent = zca_persist_01=>agent.
   try.
      agent->get_persistent( exporting i_sno    = im_sno
         i_sname  = im_sname
         receiving result = students ).
      sname = students->get_sname( ).
      sno   = students->get_sno( ).
      mark1 = students->get_mark1( ).
      mark2 = students->get_mark2( ).
      if r1 eq 'X'.
         output( ).
      endif.
   CATCH CX_OS_OBJECT_NOT_FOUND .
   MESSAGE 'Object doesn''t exists' TYPE 'I' DISPLAY LIKE 'E'.
endtry.
endmethod.                    "fetch_persistent
method output.
   write:/ sno,
      sname,
      mark1,
      mark2.
endmethod.                    "output
method create_persistent.
   fetch_persistent( exporting im_sname = im_sname
      im_sno = im_sno ).
   try.
      agent->create_persistent( exporting i_mark1 = im_mark1
         i_mark2 = im_mark2
         i_sname = im_sname
         i_sno   = im_sno
         receiving result = students ).
      commit work.
      write 'Object Created'.
   CATCH CX_OS_OBJECT_EXISTING .
   MESSAGE 'Object already exists' TYPE 'I' DISPLAY LIKE 'E'.
endtry.
endmethod.  "create_persistent
method delete_persistent.
fetch_persistent( exporting im_sname = im_sname
    im_sno = im_sno ).
try.
    agent->delete_persistent( exporting i_sname = im_sname
        i_sno   = im_sno ).
    commit work.
    write 'Object Deleted'.
    CATCH CX_OS_OBJECT_NOT_EXISTING .
    MESSAGE 'Object doesn''t exists' TYPE 'I' DISPLAY LIKE 'E'.
endtry.
endmethod.  "delete_persistent
endclass.  "lcl_class1 IMPLEMENTATION

data ref_class1 type ref to lcl_class1.
*---------------------------------------------------------------------
*          Load-of-Program                                          *
*---------------------------------------------------------------------
load-of-program.
    tit1 = text-001.
    tit2 = text-001.
*---------------------------------------------------------------------
*          Start-of-Selection                                      *
*---------------------------------------------------------------------
start-of-selection.
    create object ref_class1.
    if r1 eq 'X'.
        ref_class1->fetch_persistent( exporting im_sno = sno
            im_sname = sname ).
    elseif r2 eq 'X'.
        ref_class1->create_persistent( exporting im_sno = sno
            im_sname = sname
            im_mark1 = mark1
            im_mark2 = mark2 ).
    else.
        ref_class1->delete_persistent( exporting im_sno = sno
            im_sname = sname ).
    endif.
Persistent Objects: Using GUID Object Identity

Objective
To Store the attributes of the Objects persistently in the database.

Persistent Object’s Identity
Every Persistent Object has a unique identity with which it can be accessed. There are 2 types of Object identity

1. Business Key
2. GUID( Global Unique Identifier )

For Persistent Objects using Business key Identity please check my previous article, “Persistent Objects: Using Business Key identity”

This article will focus only on Persistent Object using GUID.

Step: 1 -> Create a database table
This table should contain a key field of type OS_GUID.

Step: 2 -> Create a Persistent Class
Go to Class Builder (tcode SE24)
Give persistent class name for eg ZCL_PERSIST_02 and hit the create button
In the next screen select the class type as Persistent Class and then hit Save Button.

Step: 3 -> Persistent Mapping or Mapping

Goto->Persistence Representation
Give the table name. For eg ZSTUDENT02 and hit the enter button.

Table fields appear in the lower half of the tool screen.
Double Click the table field

Press the upward arrow button ▲
Add the remaining fields as well. Screen looks like this now. Press Save Button

Activate the Class. Press the Yes Button to activate the class actor as well.
Unlike Business Key, GUID is not an attribute of the Persistent Class.

Step: 4 -> Write a Program to create / fetch / delete the Persistent Object

Our Program Selection-Screen looks like below
Here I am creating a new student. Specify the value and hit the execute button.

**Output:**

Go to SE16 and check the entries

<table>
<thead>
<tr>
<th>GUID</th>
<th>Student Number</th>
<th>Student Name</th>
<th>Mark</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>0122FA71F0B47646A8B0D7D8C06B0D6CFA</td>
<td>1000</td>
<td>ABDUL</td>
<td>99</td>
<td>99</td>
</tr>
</tbody>
</table>

Source Code

```
*&-------------------------------------------------------------------*
*& Report  Z_PERSISTENT_GUID
*&
*&-------------------------------------------------------------------*
*&Author : Abdul Hakim
*&Development Language: ABAP
*&System Release: SAP Netweaver 2004
*&Title: Persistent Object using GUID Object Identity!!
```
REPORT  Z_PERSISTENT_GUID.
selection-screen begin of block b1 with frame title tit1.
parameters: sno   like zstudent02-sno,
        sname like zstudent02-sname,
        mark1 like zstudent02-mark1,
        mark2 like zstudent02-mark2,
        guid  like zstudent02-guid.
selection-screen end of block b1.
selection-screen begin of block b2 with frame title tit2.
parameters: r1 radiobutton group rad1,
        r2 radiobutton group rad1,
        r3 radiobutton group rad1.
selection-screen end of block b2.
data: agent type ref to zca_persist_02,
        students type ref to zcl_persist_02.
data: result1 type ref to object,
        result2 type ref to zcl_persist_02.
load-of-program.
tit1 = text-001.
tit2 = tit1.
at selection-screen.
if ( r2 eq 'X' ).
if sno is initial or sname is initial.
MESSAGE 'Enter the values in Sno/Sname fields'
        TYPE 'E' DISPLAY LIKE 'E'.
endif.
endif.
start-of-selection.
agent = zca_persist_02=>agent.
if r1 eq 'X'.
TRY.
    CALL METHOD AGENT->IF_OS_CA_PERSISTENCY~GET_PERSISTENT_BY_OID
        EXPORTING
            I_OID  = guid
        RECEIVING
            RESULT = result1.
        result2 ?= result1.
sno = result2->get_sno( ).
sname = result2->get_sname( ).
mark1 = result2->get_mark1( ).
mark2 = result2->get_mark2( ).
write:/ sno,
sname, mark1, mark2.

CATCH CX_OS_OBJECT_NOT_FOUND.
* CATCH CX_OS_CLASS_NOT_FOUND.
MESSAGE 'Object doesn''t exist' TYPE 'I' DISPLAY LIKE 'E'.
ENDTRY.

elseif r2 eq 'X'.
TRY.

CALL METHOD AGENT->CREATE_PERSISTENT
EXPORTING
  I_MARK1 = mark1
  I_MARK2 = mark2
  I_SNAME = sname
  I_SNO = sno
RECEIVING
  RESULT = students.
commit work.
write 'Object Created'.
CATCH CX_OS_OBJECT_EXISTING.
MESSAGE 'Object already exists' TYPE 'I' DISPLAY LIKE 'E'.
ENDTRY.
else.
TRY.

CALL METHOD AGENT->IF_OS_CA_PERSISTENCY~GET_PERSISTENT_BY_OID
EXPORTING
  I_OID = guid
RECEIVING
  RESULT = result1.

CATCH CX_OS_OBJECT_NOT_FOUND.
* CATCH CX_OS_CLASS_NOT_FOUND.
MESSAGE 'Object doesn''t exist' TYPE 'I' DISPLAY LIKE 'E'.
ENDTRY.
result2 ?= result1.
TRY.

CALL METHOD AGENT->IF.OS_FACTORY~DELETE_PERSISTENT
EXPORTING
  I_OBJECT = result2.
commit work.
write 'Object Deleted'.
CATCH CX_OS_OBJECT_NOT_EXISTING.
MESSAGE 'Object doesn''t exist' TYPE 'I' DISPLAY LIKE 'E'.
ENDTRY.
endif.
Implementing Persistent Service using Transaction Service

Transaction Service is an object-oriented wrapper of SAP LUW.

In this article we will discuss how we can implement Persistent Service using Transaction Service

**Step: 1**

Go to Class Builder and create a class.

**Define methods.**

![Class Builder: Change Class ZCL_TX_SERVICE_01](image)

**Step: 2**

**Implement the methods**

method HANDLE_TRANSACTION.

    data: tx type ref to if_os_transaction,
           tx_manager type ref to if_os_transaction_manager.

    tx_manager = cl_os_system->get_transaction_manager().
    tx = tx_manager->create_transaction().

    try.
        tx->start().
        fetch_persistent().
        tx->end().
        catch cx_os_error.
        tx->undo().
    endtry.

e ndmethod.
Step: 3

Create OO transaction.

Go to Tcode SE93. Give Tcode name for eg Z_TX and hit the create button

![Maintain Transaction](image)

**Step: 4** Select OO transaction
Step: 5

Specify the Class Name and the Method Name. Also select the OO transaction Model check box. Finally Save.

Step: 6
Execute the transaction Z_TX

**Step: 7**

Go to SE16 and check the table entries

![Data Browser: Table ZSTUDENT01 Select Entries](image-url)

<table>
<thead>
<tr>
<th>Student Number</th>
<th>Student Name</th>
<th>Mark</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>ABDUL</td>
<td>01</td>
<td>02</td>
</tr>
</tbody>
</table>
## Binding in ABAP Object Oriented Programming

### Some basic terminologies

#### 1.1 ABAP Objects

ABAP objects is the term given to object oriented programming done in ABAP. This programming model unites data and functions. OO ABAP is built on existing ABAP language. ABAP objects are run in same environment as the normal ABAP programs. OO ABAP is part of ABAP since R/3 release 4.0.

#### 1.2 Class

Class is a prototype that defines data and the behaviour common to all the objects of certain kind. Here methods provide the behaviour. We can say classes describe objects.

Classes can be declared either globally or locally. Global classes can be declared using transaction SE24. Local classes are declared in an ABAP program (reports etc).

#### 1.3 Objects

It signifies the real world. Technically we can say objects are instances of a class. We can create any number of objects from a class template. All the objects created have unique identity and each contain different set of attributes. Objects we create in a program exist only till the program exists.

#### 1.4 Encapsulation

Through encapsulation we restrict the visibility of attributes and methods in the object.

There are three levels of visibility in OO ABAP.

- Public
- Protected
- Private

#### 1.5 Polymorphism

The name of method is same but they behave differently in different classes. It means implementation of method (i.e. body of the method) is different in different classes. It can be achieved in two different ways in OO ABAP.

- Interfaces
- Overriding methods or redefining methods in each class after inheritance
1.6 Inheritance

In OO ABAP we use an existing class to derive a new class (child class). The new class contains the attributes of the parent class (derives according to the visibility of attributes and methods) in addition to new attributes and methods added to it. The child class derives all the attributes and methods declared in parent class as public visibility. The child class cannot inherit private members. The protected members in parent class are derived in child class but their visibility changes to private.

1.7 Interfaces

Interfaces are similarly defined as classes. They also contain attributes and methods. But interfaces do not have implementation part. Their methods are implemented in the class that implements the interface.

So we provide different implementation to the methods defined in the interface in different class that implements that interface. This way polymorphism is achieved.

1.8 Binding in Object Oriented languages

It is the concept which determines the object’s method to be invoked based on the signature or parameters provided. It is important in case of invoking a method, which is redefined in subsequent sub classes. As the level of hierarchy increases and the same method is redefined in subclasses we need to know which method is called. When this decision is made at run time it is called as Dynamic binding. When this decision is made at compile time it is known as Static binding. In Java this concept is implemented using concept of method overriding and in C++ there is concept of virtual functions. They help in achieving dynamic binding. Similarly in OO ABAP we can redefine methods and use concept of binding to invoke methods as per required.

Binding in OO ABAP

Suppose we need to redefine or override a method of a class in all sub classes inheriting from it. We can take a pointer or reference variable to the base or parent class. This parent class reference variable can take or refer to objects of the sub classes. Now to decide that which method will be called upon while we are using this parent class reference variable, we need to know about the concept of binding.

- We define a reference variable as

  \textbf{Data : obj\_a type ref to <class name>}. \\
  Here \textit{obj\_a} is a reference variable of type class \textit{<class name>}

- Creating the object

  \textbf{create object : obj\_a}. \\
  Now \textit{obj\_a} refers to an object of class \textit{<class name>
Clear obj_a.

Now the reference variable obj_a no more refers to the object of <class name>.

In this case garbage collector will come and remove the object from memory.

### 2.1 Example

Let us create a report and some local classes and look into concept of binding

Create a report zpmm_class_dynamic.

```plaintext
*-----------------------------------------------------------------
** Report  ZPMM_CLASS_DYNAMIC                                      *
**                                                                *
------------------------------------------------------------------
REPORT  ZPMM_CLASS_DYNAMIC                      .
------------------------------------------------------------------
CLASS a DEFINITION.
PUBLIC SECTION.
  methods : rise, fall.
ENDCLASS.                 "a DEFINITION
*-----------------------------------------------------------------
** CLASS a IMPLEMENTATION                                       *
*-----------------------------------------------------------------
CLASS a IMPLEMENTATION.
  METHOD rise.
    write : / 'Super class a -------- rise()'.
  ENDMETHOD.                "rise
  METHOD fall.
    write : / 'Super class a -------- fall()'.
  ENDMETHOD.                "fall
ENDCLASS.                 "a IMPLEMENTATION
*-----------------------------------------------------------------
** CLASS b DEFINITION                                             *
*-----------------------------------------------------------------
CLASS b DEFINITION inheriting from a.
  PUBLIC SECTION.
  methods : rise redefinition,
             xyz.
ENDCLASS.                 "b DEFINITION
*-----------------------------------------------------------------
** CLASS b IMPLEMENTATION                                         *
*-----------------------------------------------------------------
CLASS b IMPLEMENTATION.
  METHOD rise.
    write : / 'Child class b redefined -------- rise()'.
  ENDMETHOD.                "rise
```
METHOD xyz.
    write : '/ Child class b new method -------- xyz()'.
ENDMETHOD.
"xyz
ENDCLASS.
"b IMPLEMENTATION
*******End of Class Definition and implementations***************
***Global declaration
***Creating reference variables for the classes defined above
data :
*Reference variable of type class a
  obj_a type ref to a,
*Reference variable of type class b
  obj_b1 type ref to b,
*Reference variable of type class b
  obj_b2 type ref to b.
******************************************************************************
******************************************************************************
*                        START-OF-SELECTION
******************************************************************************
START-OF-SELECTION.
create object : obj_a,
    obj_b1,
    obj_b2.
******************************************************************************
******************************************************************************
*                        END-OF-SELECTION
******************************************************************************
END-OF-SELECTION.
call method : obj_a->fall,
    obj_a->rise,
    obj_b1->fall.

Now output of above code is:
Super class a----------fall()
Super class a----------rise()
Super class a----------fall()

We will just discuss how we got this output and what will happen when we assign subclass objects to reference variables of parent class.

2.2 Binding

We have reference variables

obj_a , obj_b1 ,obj_b2

Further we created object obj_a (refers to object of class a) and obj_b1(refers to object of class b) using create object statement.

When we assign
obj_a = obj_b1.

Then both obj_a and obj_b now refer to same object of class b.

But obj_a is reference variable of type parent class of class b.

```
  obj_x = obj_y.
```

Reference variable Object

Now when

```
  obj_a = obj_b.
```

Reference variable is of type Base Class

Object passed is of type Sub Class.

When we will use the reference variable obj_a to invoke method rise() which is overridden in sub class b, the sub class b method rise() (redefined method) is invoked.

So if we change the code below START-OF-SELECTION event and END-OF-SELECTION event in section 2.1 to check the above theory.

```
*****************************************************************
*                        START-OF-SELECTION
******************************************************************
START-OF-SELECTION.
create object : obj_a,
               obj_b1,
               obj_b2.
obj_a = obj_b1.
******************************************************************
*                        END-OF-SELECTION
******************************************************************
END-OF-SELECTION.
call method : obj_a->fall,
              obj_a->rise,
              obj_b1->fall.
```

Now output of above code is :
Super class a---------fall()
Child class b redefined---------rise()
Super class a---------fall()
2.3 Binding Check Table

I have prepared a table to check the method invoked in case of inheritance. This table is used to check the method invoked when the method is redefined in sub classes.

<table>
<thead>
<tr>
<th>Reference Variable</th>
<th>Object</th>
<th>Method Invoked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Base Class</td>
<td>Base Class</td>
</tr>
<tr>
<td>Base Class</td>
<td>Sub Class</td>
<td>Sub Class</td>
</tr>
<tr>
<td>Sub Class</td>
<td>Sub Class</td>
<td>Sub Class</td>
</tr>
</tbody>
</table>

**Note:** We can not take a reference variable of Sub Class to refer a Base class object.

\[
\text{obj}_b1 = \text{obj}_a. \text{ is not possible}
\]

We can now verify the output of code given in section section 2.1.

2.4 Important in Binding

Till now we have seen which method is called when we use reference variable of base class and pass the object of sub class. But there are some restrictions on calling methods.

\[
\text{obj}_x = \text{obj}_y.
\]

**When** \( \text{obj}_a = \text{obj}_b. \)

When reference variable is of base class i.e \( \text{obj}_a \)

And object referred by \( \text{obj}_a \) is of type subclass i.e. \( \text{obj}_b. \)

In this case base class reference variable can only call the methods which are defined there in the base class.
We can not invoke the new method defined in the class b `xyz()` using base class `obj_a` reference variable.

```
obj_a = obj_b.
```


In this case we can only call

```
call method : `obj_a->fall`,
```

obj_a->rise.

When we call `obj_a->fall`, it will call the method of base class since it is not redefined in sub class b.

When we call `obj_a->rise`, it will call the method of sub class since it is redefined in sub class b. For this we can use the table of section 2.3.

### 2.5 Output of section 2.1

We can now verify the output of code given in section 2.1 from the table described in section 2.3

We will just see the START-OF-SELECTION and END-OF-SELECTION events below from section 2.1

```
******************************************************************
*                        START-OF-SELECTION
******************************************************************
START-OF-SELECTION.
create object : `obj_a`,
    `obj_b1`,
    `obj_b2`.
```

```
******************************************************************
*                        END-OF-SELECTION
******************************************************************
END-OF-SELECTION.
call method : `obj_a->fall`,
    `obj_a->rise`,
    `obj_b1->fall`.
```

Now output of above code is :

Super class a----------fall()
Super class a----------rise()
Super class a----------fall()

Here `obj_a` refers to base class object so it only calls base class methods rise() and fall().

Since method `fall()` is not redefined in class b and is just inherited from class a , so when we call `obj_b1->fall`, the base class method is invoked.
Understanding "ABAP Unit"

Introduction:

It is a best practice to modularize our programs as much as we can for better programming. If we want to check one particular module like subroutines, function modules or classes for bugs then we can do it using ABAP Unit. ABAP Unit is a tool for unit testing of ABAP programs.

How to write these tests:

ABAP unit is based on ABAP objects. The global class CL_AUNIT_ASSERT contains methods which can be used for testing. Tests are implemented in local classes. Inside the local class the necessary method from the global class can be called for testing. These test classes can be written inside the program for which the test is to be done. It will not affect our production code in anyways.

Difference between Ordinary class and Test class:

Both the test class and test method should have FOR TESTING addition.

Ex:

```
CLASS mytest DEFINITION FOR TESTING.
  PRIVATE SECTION.
  METHODS mytest FOR TESTING.
ENDCLASS.
```

Methods in CL_AUNIT_ASSERT for Testing:

- ASSERT_EQUALS
- ASSERT_DIFFERS
- ASSERT_BOUND
- ASSERT_NOT_BOUND
- ASSERT_INITIAL
- ASSERT_NOT_INITIAL
- ASSERT_CHAR_CP
- ASSERT_CHAR_NP
- ASSERT_EQUALS_F
- FAIL
- ABORT

ASSERT_EQUALS - Checks the equality of two data objects.

ASSERT_DIFFERS - Checks for the difference of two data objects.
ASSERT_BOUND - checks for the validity of the reference of a reference variable.

ASSERT_INITIAL - checks whether the reference of a reference variable is invalid.

ASSERT_NOT_INITIAL - checks whether the data object is not having its initial value.

ASSERT_SUBRC - checks for the specific value of SY-SUBRC.

**ASSERT_EQUALS:**

ASSERT_EQUALS is one of the methods in the class CL_AUNIT_ASSERT. This method can be used for checking equality of two data objects.

The parameters of the method:

- **ACT** - Actual result
- **EXP** - Expected Result
- **MSG** - Message to be displayed in the result
- **LEVEL** - Error level (Tolerable/Critical/fatal)
- **QUIT** - If the test fails, flow level is controlled using this
  
  (NO/METHOD/CLASS/PROGRAM)
- **TOL** - Tolerance level for F

**Levels:**

- 0 - Tolerable
- 1 - Critical
- 2 - Fatal

**Quit:**

- No (0) – It will continue the current test Method.
- Method (1) – It will interrupt the current test method
- Class (2) – It will interrupt the current test class.
- Program (3) – abandon execution of all test classes for the tested program.

**Tolerance:**

If the tolerance limit specified is exceeded then error is shown.

Ex:
Actual result – 24.

Expected Result – 25.

\[ Tolerance = 0.9999. \]

\[ Difference = \text{Expected Result} - \text{Actual result}. \]
\[ = 1 > \text{tolerance}. \]

Therefore displays an error.

**Example Program:**

Let us consider an example for ABAP unit test using the method `ASSERT_EQUALS` to check the equality of two data objects. In this program, we have two methods `divide` and `factorial` in a local class `MATH`. We want to test the `factorial` method. So we have created one class and one method `MYTEST` for testing. In the test method implementation we have called the `factorial` method and so the data object `RESULT` is populated. Now we are going to compare the actual data object (`RESULT`) with the expected result. For that we are calling the `ASSERT_EQUALS` from the global class passing the expected result.

```abap
*----------------------------------------------------------------------*
*       CLASS math DEFINITION                                         *
*----------------------------------------------------------------------*
CLASS math DEFINITION.
PUBLIC SECTION.
METHODS divide
  IMPORTING opr1 TYPE i
  opr2 TYPE i
  EXPORTING result TYPE f
  RAISING cx_sy_arithmetic_error.
METHODS factorial
  IMPORTING n TYPE i
  RETURNING value(fact) TYPE i.
ENDCLASS.  "math DEFINITION
*----------------------------------------------------------------------*
*       CLASS math IMPLEMENTATION                                    *
*----------------------------------------------------------------------*
CLASS math IMPLEMENTATION.
METHOD divide.
  result = opr2 / opr1.
ENDMETHOD.  "divide
METHOD factorial.
  fact = 1.
  IF n = 0.
    RETURN.
  ELSE.
    DO n TIMES.
      fact = fact * sy-index.
  ENDDO.
```
START-OF-SELECTION.
DATA w_obj TYPE REF TO math.
DATA exc  TYPE REF TO cx_sy_arithmetic_error.
DATA res  TYPE f.
DATA result TYPE i.
DATA text TYPE string.
CREATE OBJECT w_obj.
TRY.
  w_obj->divide( EXPORTING opr1 = 32 opr2 = 4 
    IMPORTING result = res ).
  WRITE : res.
  text = res.
  CATCH cx_sy_arithmetic_error INTO exc.
  text = exc->get_text( ).
  MESSAGE text TYPE 'I'.
ENDTRY.
CREATE OBJECT w_obj.
COMPUTE result = w_obj->factorial( 4 ).
WRITE :/ 'The result for factorial is:',result.
*
* \  CLASS mytest DEFINITION
* \----------------------------------------------------------
* \CLASS mytest DEFINITION "#AU Risk_Level Harmless 
\FOR TESTING. "#AU Duration Short
PRIVATE SECTION.
METHODS mytest FOR TESTING.
ENDCLASS.    "mytest DEFINITION
* \----------------------------------------------------------
* \CLASS mytest IMPLEMENTATION
* \------------------------------------------------------------------
* \CLASS mytest IMPLEMENTATION.
METHOD mytest.
  CREATE OBJECT w_obj.
  result = w_obj->factorial( 4 ).
  cl_aunit_assert=>assert_equals( act    = result 
    exp    = '24' 
    msg    = 'Factorial Not calculated Correctly' 
    level  = '0' 
    quit   = '2' 
    tol    = '0.999' 
  ).
ENDMETHOD.  "mytest
ENDCLASS.     "mytest IMPLEMENTATION
Executing Unit Tests:

For program,

Program -> Test -> Unit Test.

For class,

Class -> Unit Test.

For Function Module,

Function Module -> Test -> Unit Test.

Result of Unit Test:

If both the actual and the expected result is same, then Unit test does not find any errors. In that case one message will be displayed on status bar like,

Unit tests processed successfully, 1 programs, 1 classes, 1 methods

If it finds errors then a result will be displayed as follows:

The task is displayed in a tree structure with a Program name, Class name and method name. Both the expected and the actual results can be seen in the Unit test results. Also in
the stack it will be displaying the line number where the error occurred. By double clicking the line number we can enter into the source code.

**ABAP Unit results in Code Inspector:**

We can see the ABAP unit results in code inspector. While creating the variant, check for the ABAP unit in Dynamic check.

![Code Inspector: Check Variant](image)

In the Code inspector results we can check for the ABAP unit errors, warnings and informations.

![Code Inspector: Results from ZOBJ 001 SAPDEVV02](image)
**Demo on "Narrow Casting"**

**Definition:** The assignment of a subclass instance to a reference variable of the type "reference to super class" is described as a narrowing cast, because you are switching from a more detailed view to a one with less detail. It is also called as up-casting.

**Use of narrowing casting:**

A user who is not interested in the finer points of cars, trucks, and busses (but only, for example, in the fuel consumption and tank gauge) does not need to know about them. This user only wants and needs to work with (references to) the lcl_vehicle(super class) class. However, in order to allow the user to work with cars, busses, or trucks, you generally need a narrowing cast.

**Principle of narrowing casting:**

1. In narrowing casting the object which is created with reference to the sub class is assigned to the reference of type super class.
2. Using the super class reference it is possible to access the methods from the object which are only defined at the super class.
3. This access is also called as generic access as super class is normally called as general class.

**Example:**

Super class: vehicle (contains general methods)
Sub class: truck (contains more specific methods)

Here method4 is the specific for the sub class and remaining methods are inherited from the super class.

Now create the object with reference to the subclass.

1. Declare a variable with reference to the subclass.

   DATA: REF_TRUCK TYPE REF TO TRUCK.

2. Create object with this reference.

   CREATE OBJECT REF_TRUCK.

Narrowing cast:

1. Declare a variable with reference to the super class.

   DATA: REF_VEHICLE TYPE REF TO VEHICLE.

2. Assign the object reference (REF_TRUCK) to REF_VEHICLE.

   REF_VEHICLE = REF_TRUCK.
Accessing methods using super class reference.

1. By the super class reference (REF_VEHICLE) it is possible to access all the methods which are defined at the super class but the implementations are taken from the sub class.

2. If any method is redefined at the sub class then that method’s implementation which exist at the sub class is taken in to consideration.

E.g. assume that ‘method2’ is redefined at the sub class.

When this method is accessed using the super class reference

Like:

Call method REF_VEHICLE->method2.

Here we can access the implementation which exists at the sub class but not from the super class.

3. It is not possible to access the methods which only defined in the sub class using the super class reference.

E.g. Method4 is not accessed using reference REF_VEHICLE.

Call method REF_VEHICLE-> Method4.

This is wrong convention.

Demo for narrowing casting:

Go to transaction SE38.
REPORT znarrowing_cast1.

CLASS cl_super DEFINITION.

PUBLIC SECTION.

METHODS: add IMPORTING f_a TYPE i
          f_b TYPE i
          EXPORTING f_c TYPE i.

ENDCLASS.  "CL SUPER DEFINITION

CLASS cl_super IMPLEMENTATION.

METHOD add.

  f_c = f_a + f_b.

ENDMETHOD.  "ADD

ENDCLASS.  "CL SUPER IMPLEMENTATION
CLASS CL_SUB DEFINITION

PUBLIC SECTION.

METHODS: ADD REDEFINITION.

METHODS: sub IMPORTING f_a TYPE i
         f_b TYPE i
         EXPORTING f_c TYPE i.

ENDCLASS. "CL_SUB DEFINITION"

CLASS CL_SUB IMPLEMENTATION

METHOD ADD.

  F_C = F_A + F_B + 1.

ENDMETHOD.

METHOD sub.

  f_c = f_a - f_b.

ENDMETHOD. "SUB"

ENDCLASS. "CL_SUB IMPLEMENTATION"
Now execute (F8):

```
P_A   P_B
3     2
```

Result:

```
Demo on narrowing casting
6
```
Abstract Classes and Methods in Object Oriented Programming

**Abstract Class:** Classes which contain one or more abstract methods or abstract properties, such methods or properties do not provide implementation. These abstract methods or properties are implemented in the derived classes (Sub-classes).

Abstract classes does not create any instances to that class objects.

**Use of Abstract class:**

We can define some common functionalities in Abstract class (Super-class) and those can be used in derived classes (Sub classes).

**Step-by-Step Approach to create Abstract classes and Methods**

TCODE: SE24

Enter the name of class as 'Z_DEMO_ABS_CLASS' and press Create Button

A pop-up window is displayed, then select "Class" radio button and

Press enter

It will go to the next screen.
Here you enter the Description of the class and then select "Abstract" from Instantiation Drop down list to define the class an abstract class,

Then click on save button

Go to the "Attributes" tab,

Enter the Attribute name, Level, Visibility, Type and Description as shown in the screen shot.

Go to Methods tab,

Enter Method name, Level, Visibility and Description as shown in the below screen shot
Double click on the Method name "AREA"; it goes to method Implementation screen.

Go to Menu path, then Goto -> Method definition
To define method "AREA" as an abstract method,

Go to "Attributes" tab, check the check box "Abstract"

When you click on the "Abstract" check box, pop-up window is displayed,

then press ☑ button.

Then press "Change" button.
A successful message is displayed like "Method changed successfully"

Creating Sub Class:

TCode: SE24

Enter the name of class as 'Z_DEMO_ABS_SUB_CLASS' and press Create Button to create sub class
A pop-up window is displayed, then select "Class" radio button and Press enter.

It goes to next screen, here you enter the Description of the class and then

Select the inheritance button, to inherit the super class then press button.
Enter the Super class name as "Z_DEMO_ABS_CLASS", which is being created earlier and press **Save** button.

The Attributes and methods defined in the super class will automatically come into the sub class. See the below screen shots.
Go to the Methods tab, select the "AREA" method and click on "Redefine" button.

If you are not redefine the method and trying to activate the class, it gives syntax error.

Here you can write the code.
Write the code In between Method and End Method

    Method AREA
        ....
    Endmethod

Write the below code

method AREA

* Local Data Declarations
DATA: lv_count TYPE i,
     lv_res TYPE i.

* initialize Count value to '1'
lv_count = '1'.

DO 10 TIMES.
    IF lv_count <= '10'.
        lv_res = v_num * lv_count.
    ENDIF.

* Display the multiplication table for a Given Number
WRITE: / v_num, 'x', lv_count, '=' lv_res.

* Increment Count value
lv_count = lv_count + 1.
ELSE.
    EXIT.
ENDIF.
ENDDO.

* Clear variable
CLEAR: v_num.

endmethod
Then save and activate the class and method.

Finally execute the class \( \text{(F8)} \) \( \text{[Image: Class Builder: Change Class Z_DEMO_ABS_SUB_CLASS]} \)

It goes to below screen
Enter the number under V_NUM as "6" and press execute button 🔴.

The output will be displayed like below.
Final Classes and Methods in Object Oriented Programming

**Final Class:** A class that is defined as final class can not be inherited further. All Methods of a final class are inherently final and must not be declared as final in the class definition. Also, a final method can not be redefined further. If only a method of a class is final then that class can be inherited but that method cannot be redefined.

**Use of final class:**
If you don't want anyone else to change or override the functionality of your class then you can define it as final. Thus no one can inherit and modify the features of this class.

**Step-by-Step Approach to create Final classes and Methods**

**Final Class:**
TCode: SE24

Enter the name of class as 'Z_DEMO_FINAL_CLASS' and press Create Button

A pop-up window is displayed, then select "Class" radio button and

Press enter
Enter the Description of the class and select the check box "Final" to define the class as Final class, and then press enter.

Go to the "Attributes" tab,

Enter the Attribute name, Level, Visibility, Type and Description as shown in the screenshot.
Go to Methods tab,

Enter Method name, Level, Visibility and Description as shown in the below screen shot

Double click on the Method name "METH"; it goes to method Implementation screen.

Here write the code.

```
METHOD meth.
  
  * Local Data Declarations
  DATA: lv_sum  TYPE i VALUE '1',
       lv_val1 TYPE i VALUE '0',
       lv_val2 TYPE i VALUE '0'.

  WRITE: '/Fibonacci Series'.

  WHILE lv_sum <= number.
    WRITE: lv_sum.
    lv_val1 = lv_val2.
    lv_val2 = lv_sum.
    lv_sum = lv_val1 + lv_val2.
  ENDWHILE.

ENDMETHOD.
```

Write the following code in method meth
Method meth

* Local Data Declarations
DATA: lv_sum TYPE i VALUE '1',
    lv_val1 TYPE i VALUE '0',
    lv_val2 TYPE i VALUE '0'.

WRITE: / 'Fibonacci Series'.

WHILE lv_sum <= number.
  WRITE: lv_sum.
  lv_val1 = lv_val2.
  lv_val2 = lv_sum.
  lv_sum = lv_val1 + lv_val2.
ENDWHILE.

Endmethod.

Then save and activate the class and method.

Finally execute the class by pressing (F8) button.

It goes to below screen, then enter value under "NUMBER" as "19" and Press execute button.

Test Class Z_DEMO_FINAL_CLASS

The output will be displayed like below.
**Final Method:**

**a) Creating Super Class:**

TCode: SE24

Enter the name of class as 'Z_DEMO_SUP_CLASS' to create super class and then press "Create" Button

A pop-up window is displayed, then select "Class" radio button and Press enter
Enter the Description of the class and then press enter.

Go to the "Attributes" tab,

Enter the Attribute name, Level, Visibility, Type and Description as shown in the screen shot.

Go to Methods tab,

Enter Method name, Level, Visibility and Description as shown in the below screen shot.
Double click on the Method name "VOLUM"; it goes to method Implementation screen. As shown below

To define method "VOLUM" as a Final method,

Go to Menu path, then Goto -> Method definition
Pop-up window is displayed

Go to "Attributes" tab, check the check box "Final" and then press "Change" button.

<table>
<thead>
<tr>
<th>Class</th>
<th>Z_DEMO_SUP_CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>VOLUM</td>
</tr>
<tr>
<td>Description</td>
<td>To calculate Volum</td>
</tr>
</tbody>
</table>

![Change Method VOLUM](image)

A successful message is displayed like "Method changed successfully"

Method changed successfully

Write the below code in Method volume
METHOD volum.
* Local Data Declarations
DATA: lv_vol TYPE i.

lv_vol = length * width * height.
WRITE: 'Volume of a Rectangle:', lv_vol.
* Clear variable
CLEAR: lv_vol.
ENDMETHOD.

Then save and activate the class and method.

b) Creating Sub Class:
TCode: SE24

Enter the name of class as 'Z_DEMO_SUB_CLASS' and press Create Button to create sub class
A pop-up window is displayed, then select "Class" radio button and

Press enter

Enter the Description of the class and then select the inheritance button  , to inherit the super class.

Enter the Super class name as "Z_DEMO_SUP_CLASS", which is being created earlier and press "Save" button.

The Attributes and methods defined in the super class will automatically come into the sub class.

If you try to redefine or modify the super class method "VOLUM", go to the Methods tab, select the "VOLUM" method and click on

"Redefine" button ,
It gives a message like below and not allowed to redefine or modify the method in sub class.

Method VOLUM is final and therefore cannot be redefined

The method implementation "VOLUM" can be used in both super class "Z_DEMO_SUP_CLASS" and sub class "Z_DEMO_SUB_CLASS".

Execute the sub class "Z_DEMO_SUB_CLASS" by pressing (F8) button.

It goes to below screen, then enter values under "LENGTH, HEIGHT and WIDTH" as "2, 3 and 4" and Press execute button.

Test Class Z_DEMO_SUB_CLASS
Same as sub class, you can also execute the super class "Z_DEMO_SUP_CLASS", the same output will be displayed.
**Redefining methods in subclass**

**Definition:** The methods of the super class can be re-implemented at the sub class.

**Purpose to redefine methods:** if the method implementation at the super class is not satisfies the requirement of the object which is created with reference to the sub class.

**Principles of redefining methods:**

1. The REDEFINITION statement for the inherited method must be in the same SECTION as the definition of the original method.
2. If you redefine a method, you do not need to enter its interface again in the subclass, but only the name of the method.
3. In the case of redefined methods, changing the interface (overloading) is not permitted; exception: Overloading is possible with the constructor.
4. Within the redefined method, you can access components of the direct super class using the SUPER reference.
5. The pseudo-reference super can only be used in redefined methods.

**Demo program for redefining method:**

Go to transaction SE38:

Give any name for the program.
REPORT zinheritance_redefinition1.

CLASS cl_super DEFINITION.

PUBLIC SECTION.

METHODS: add IMPORTING f_a TYPE i
          f_b TYPE i
          EXPORTING f_c TYPE i.

ENDCLASS.       "CL SUPER DEFINITION"

CLASS cl_super IMPLEMENTATION.

METHOD add.

  f_c = f_a + f_b.

ENDMETHOD.      "ADD"

ENDCLASS.       "CL SUPER IMPLEMENTATION"
After execution (F8):

```
P_A
5
P_B
6
```

Result:
Demo for redefinition of method

Demo for use of super keyword in redefinition:

Go to transaction SE38.

```plaintext
REPORT zinheritance_redefinition_sup.

CLASS cl_super DEFINITION.

PUBLIC SECTION.
METHODS: add IMPORTING f_a TYPE i,
         f_b TYPE i
         EXPORTING f_c TYPE i.
ENDCLASS.  "CL SUPER DEFINITION

CLASS cl_super IMPLEMENTATION.

METHOD add.

f_c = f_a + f_b.
ENDMETHOD.  "ADD
ENDCLASS.  "CL SUPER IMPLEMENTATION
```
CLASS cl_sub DEFINITION INHERITING FROM cl_super.

PUBLIC SECTION.

METHODS: add REDEFINITION.

ENDCLASS.  "CL_SUB DEFINITION

CLASS cl_sub IMPLEMENTATION.

METHOD add.

*Use of SUPER KEY.
  DATA: gv_var1 TYPE i.
  DATA: gv_var2 TYPE i.
  DATA: gv_var3 TYPE i.
  gv_var1 = 10.
  gv_var2 = 20.

  CALL METHOD super->add
  EXPORTING
      f_a = gv_var1
      f_b = gv_var2
  IMPORTING
      f_c = gv_var3.
  f_c = gv_var3 + f_a + f_b.

ENDMETHOD.  "ADD

ENDCLASS.  "CL_SUB IMPLEMENTATION
START-OF-SELECTION.

PARAMETERS: p_a TYPE i,

    p_b TYPE i.

DATA: gv_add TYPE i.
DATA: gv_sub TYPE i.

DATA: ref1 TYPE REF TO cl_sub.

CREATE OBJECT ref1.

CALL METHOD ref1->add

    EXPORTING
    f_a = p_a
    f_b = p_b

    IMPORTING
    f_c = gv_add.

WRITE:/ gv_add.

After execution (F8):

<table>
<thead>
<tr>
<th>P_A</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_B</td>
<td>5</td>
</tr>
</tbody>
</table>

Result:

Use of super keyword

39
Handling Data in Excel In-place Display Using BDS

The article demonstrates data handling in excel in-place display using BDS with the help of a program. The demo program maintains the entries in a database table through an excel in-place display.

OVERVIEW

MS Excel is the conventional way of storing and maintaining data. Sometimes, user prefers to display report output in specific MS Excel templates; which contain logos, user specific table formats, engineering drawings, diagrams and macros. The data modified manually in these excel templates may be again transferred to SAP for processing.

Excel integration is required due to various reasons like avoiding user training on newly developed custom programs and screens, use existing data templates, data integration with legacy system.

BDS (Business Document Services) is a convenient option for excel integration as user specific MS Excel templates can be stored in it. These templates can be called in an ABAP program at runtime to display data. Also, the data modified by the user in MS Excel can be read into ABAP program for processing.

The functionality will be demonstrated through a demo program. The program will display the content of a custom table in excel in-place display. The user can change the non key fields displayed and the modified contents will be updated to the table after validation.

1. Defining a BDS Class

A custom BDS class can be defined through transaction SBDSV1 as described below. An existing BDS class can be used, unless the user wants a separate class for a specific application.

In SBDSV1, Go to ‘NEW ENTRIES’.
Enter the ‘Class name’, ‘Class type’ as ‘Other objects(OT)’, ‘Log Level’ as required and rest of the parameters should filled as shown below.

2. Uploading MS Excel Template

Design a template as per user requirement in MS Excel. You can embed all static objects/data to be displayed such as logos, drawings, headers etc in the template, except the area, where the data will be filled at runtime.

A sample template has been created as shown below.
Now, the MS Excel template needs to be uploaded to BDS using transaction OAOR under a class. Enter any existing Class Name, Class Type as ‘OT’ and Object Key in the selection screen of OAOR. Object key is like a sub folder, which is used to distinguish different sets of documents stored under a class. Any value can be entered to define an object key in OAOR. But to access a document, the same object key must be keyed in, in which it was stored initially.

Now, go to ‘Create’ tab and double click on table template. It will show a pop up to upload the MS Excel template.
Enter the ‘Description’ for the table template after uploading.

The uploaded table template can be attached to a transport request as well.
3. Code to Handle Data in Excel In-place Display

The program will maintain a custom table YSM_AGENTS, which has the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Key</th>
<th>initi</th>
<th>Data element</th>
<th>Data Ty</th>
<th>Length</th>
<th>Decim</th>
<th>Short Des</th>
</tr>
</thead>
<tbody>
<tr>
<td>MANDT</td>
<td>✔️</td>
<td>✔️</td>
<td>MANDT</td>
<td>CLNT</td>
<td>3</td>
<td>0</td>
<td>Client</td>
</tr>
<tr>
<td>AGENTID</td>
<td>✔️</td>
<td></td>
<td>CHAR</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>Agent Id</td>
</tr>
<tr>
<td>NAME</td>
<td></td>
<td></td>
<td>CHAR</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>Name</td>
</tr>
<tr>
<td>EMAIL</td>
<td></td>
<td></td>
<td>CHAR</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>Email</td>
</tr>
</tbody>
</table>

Initially, the program will display the table contents of YSM_AGENTS in the excel template uploaded in BDS. The user should be able to modify only the non key fields of the table filled with color green. So, we need to protect the whole worksheet except a range or window, which will contain editable fields NAME & EMAIL. The user will not be able to modify anything else except these fields.

Also, the email entered will be validated. If an invalid email id is entered, error message will be displayed with the cell to be corrected filled with color red.
Create a screen ‘0100’ and a custom control ‘EXCEL’ in it to display the excel document in-place. Also, activate the BACK, EXIT, CANCEL, SAVE options in GUI status.

*&---------------------------------------------------------------------*  
*& Report YSM_TESTS5*  
*&---------------------------------------------------------------------*  
*& Demo program for displaying table data in a specific excel template*  
*& using BDS. Also, reads the contents modified by user again into ABAP*  
*& program after validations and updates the table.*  
*&---------------------------------------------------------------------*  
REPORT ysm_test5.

***********************************************************************
* Data Declaration***********************************************************************
* Custom Table With 3 fields  
* AGENTID (KEY)  
* NAME  
* EMAIL  
TABLES: ysm_agents.

TYPES: BEGIN OF t_agents,  
agentid TYPE ysm_agents-agentid,  
name TYPE ysm_agents-name,  
email TYPE ysm_agents-email,  
END OF t_agents.

DATA: int_agents TYPE TABLE OF t_agents,  
wf_entries TYPE i.

TYPE-POOLS: soi,  
sbdst.

DATA: r_document TYPE REF TO cl_bds_document_set,  
r_excel TYPE REF TO i_oi_spreadsheet,  
r_container TYPE REF TO cl_gui_custom_container,
**Selection Screen**

SELECTION-SCREEN BEGIN OF BLOCK b1 WITH FRAME.

* User will enter the agent ids to be modified

SELECT-OPTIONS: s_agent FOR ysm_agents-agentid OBLIGATORY.

* Details of table template in BDS to be entered

PARAMETERS: p_clsnam TYPE sbdst_classname DEFAULT 'YSM_TESTBDS' OBLIGATORY,
p_clstyp TYPE sbdst_classtype DEFAULT 'OT' OBLIGATORY,
p_objkey TYPE sbdst_object_key DEFAULT 'TEST' OBLIGATORY,
p_desc TYPE char255 DEFAULT 'TABLE TEMPLATE' OBLIGATORY.

SELECTION-SCREEN END OF BLOCK b1.

**Screen Logic**

*&---------------------------------------------------------------------* & Module STATUS_0100 OUTPUT
*&---------------------------------------------------------------------*

MODULE status_0100 OUTPUT.

SET PF-STATUS 'STAT100'. "Enable SAVE, BACK, EXIT, CANCEL
SET TITLEBAR 'TITLE100'. "Set title

* Get table data
PERFORM f_get_table_data.

* Open the excel template in BDS in-place
PERFORM f_open_document USING p_clsnam
  p_clstyp
  p_objkey
  p_desc.

* Display table data in the excel template
PERFORM f_dis_table_data.

* Protect the whole sheet except the editable fields
PERFORM f_protect_sheet.

ENDMODULE. " STATUS_0100 OUTPUT

*&---------------------------------------------------------------------* & Module USER_COMMAND_0100 INPUT
*&---------------------------------------------------------------------*
MODULE user_command_0100 INPUT.

CASE sy-ucomm.
  WHEN 'BACK' OR 'EXIT' OR 'CANCEL'.
    * Close document
      PERFORM f_close_document.
      LEAVE TO SCREEN 0.
      WHEN 'SAVE'.
    * Save the modified entries into database
      PERFORM f_save_document.
  END CASE.
ENDMODULE.  " USER_COMMAND_0100  INPUT

************************************************************************************
* SUBROUTINES
************************************************************************************
*&-----------------------------------------------------------------------------*
*&    Form  f_get_table_data
*&-----------------------------------------------------------------------------*
*-----------------------------------------------------------------------------*
FORM f_get_table_data .

* Get all the agents from table
SELECT  agentid
  name
  email
FROM ysm_agents
INTO TABLE int_agents
WHERE agentid IN s_agent.

IF sy-subrc NE 0.
  MESSAGE 'No Agent Details Found' TYPE 'E'.
ENDIF.

* Get the no of rows to be displayed
DESCRIBE TABLE int_agents LINES wf_entries.
ENDFORM.  " f_get_table_data

*&-----------------------------------------------------------------------------*
*&    Form  f_open_document
*&-----------------------------------------------------------------------------*
*-----------------------------------------------------------------------------*
FORM f_open_document USING l_clsnam TYPE sbdst_classname
  l_clstyp TYPE sbdst_classid
  l_objkey TYPE sbdst_object_key
  l_desc  TYPE char255.

DATA: locint_signature  TYPE sbdst_signature,
       locint_uris     TYPE sbdst_uri,
locwa_signature  LIKE LINE OF locint_signature,
locwa_uris      LIKE LINE OF locint_uris.

IF NOT r_document IS INITIAL.
RETURN.
ENDIF.

* Create container control
CALL METHOD c_oi_container_control_creator=>get_container_control
IMPORTING
  control = r_control
  retcode = wf_retcode.

IF wf_retcode NE c_oi_errors=>ret_ok.
CALL METHOD c_oi_errors=>raise_message
  EXPORTING
    type = 'E'.
ENDIF.

* Initialize Custom Control
CREATE OBJECT r_container
  EXPORTING
    container_name = 'EXCEL'. "Custom Control Name

CALL METHOD r_container->init_control
  EXPORTING
    r3_application_name = 'EXCEL INPLACE BDS'
    inplace_enabled     = abap_true
    inplace_scroll_documents = abap_true
    parent              = r_container
  IMPORTING
    retcode             = wf_retcode.

IF wf_retcode NE c_oi_errors=>ret_ok.
CALL METHOD c_oi_errors=>raise_message
  EXPORTING
    type = 'E'.
ENDIF.

* Create object for cl_bds_document_set
CREATE OBJECT r_document.

* Get Document with URL
locwa_signature-prop_name  = 'DESCRIPTION'.
* Description of the table template in OAOR
locwa_signature-prop_value = l_desc.
APPEND locwa_signature TO locint_signature.

CALL METHOD r_document->get_with_url
  EXPORTING
    classname  = l_clsnam
    classtype  = l_clstyp
    object_key = l_objkey
  CHANGING
    uris        = locint_uris
    signature   = locint_signature
  EXCEPTIONS
    nothing_found = 1
    error_kpro   = 2
internal_error  = 3
parameter_error = 4
not_authorized   = 5
not_allowed      = 6.

IF sy-subrc NE 0.
  MESSAGE 'Error Retrieving Document' TYPE 'E'.
ENDIF.

READ TABLE locint_uris INTO locwa_uris INDEX 1.

CALL METHOD r_control->get_document_proxy
  EXPORTING
    document_type  = 'Excel.Sheet'
  IMPORTING
    document_proxy = r_proxy
    retcode        = wf_retcode.

IF wf_retcode NE c_oi_errors=>ret_ok.
  CALL METHOD c_oi_errors=>show_message
    EXPORTING
      type = 'E'.
ENDIF.

* Open Document
CALL METHOD r_proxy->open_document
  EXPORTING
    document_url  = locwa_uris-url
    open_inplace  = abap_true
    protect_document = abap_true "Protect Document initially
  IMPORTING
    retcode        = wf_retcode.

IF wf_retcode NE c_oi_errors=>ret_ok.
  CALL METHOD c_oi_errors=>show_message
    EXPORTING
      type = 'E'.
ENDIF.

* Get Excel Interface
CALL METHOD r_proxy->get_spreadsheet_interface
  IMPORTING
    sheet_interface = r_excel
    retcode         = wf_retcode.

IF wf_retcode NE c_oi_errors=>ret_ok.
  CALL METHOD c_oi_errors=>show_message
    EXPORTING
      type = 'E'.
ENDIF.

ENDFORM.  " f_open_document

*---------------------------------------------------------------------*  
*  Form f_dis_table_data                                           *  
*---------------------------------------------------------------------*  
* Display data in table template                                 *  
*---------------------------------------------------------------------*  
FORM f_dis_table_data .
DATA: locint_fields TYPE TABLE OF rfc_fields.

* Create a range to insert data
PERFORM f_create_range USING 9 /*Begin on 9th row
                              3 /*Begin on 3rd col
                              wf_entries "No of rows reqd
                              3 /*No of cols reqd
                              'AGENTS'. /*Range name
*-> Set Frame to the range
*# Calculation of TYP parameter
* The parameter has 8 bits
*0 Sets the left margin
*1 Sets the top margin
*2 Sets the bottom margin
*3 Sets the right margin
*4 Horizontal line
*5 Sets the left margin
*6 Thickness
*7 Thickness
* My figure will be 7 6 5 4 3 2 1 0
*   1 0 1 1 1 1 1 1
* Binary 1011 1111 stands for 191 in decimal
* Check SAP help for more info.....
  "21/b531bfe1ba11d2bdbe08009b4534c/frameset.htm

CALL METHOD r_excel->set_frame
  EXPORTING
    rangename = 'AGENTS'
    typ        = 191
    color      = 21
IMPORTING
    error      = r_error
    retcode    = wf_retcode.

IF r_error->has_failed = abap_true,
  CALL METHOD r_error->raise_message
    EXPORTING
      type = 'E'.
ENDIF.

* Get field attributes of the table to be displayed
CALL FUNCTION 'DP_GET_FIELDS_FROM_TABLE'
  TABLES
    data       = int_agents
    fields     = locint_fields
EXCEPTIONS
  dp_invalid_table = 1
  OTHERS         = 2.

IF sy-subrc <> 0.
  MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
    WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
ENDIF.

* Insert the table entries into Excel
CALL METHOD r_excel->insert_one_table
EXPORTING
  fields_table = locint_fields[] "Defn of fields
data_table = int_agents[] "Data
rangename = 'AGENTS' "Range Name
IMPORTING
  error = r_error
  retcode = wf_retcode.

IF r_error->has_failed = abap_true.
  CALL METHOD r_error->raise_message
  EXPORTING
    type = 'E'.
ENDIF.
ENDFORM. " f_dis_table_data

*&---------------------------------------------------------------------*
error = r_error
retcode = wf_retcode.

IF r_error->has_failed = abap_true.
  CALL METHOD r_error->raise_message
    EXPORTING
      type = 'E'.
  ENDIF.
ENDIF.
ENDIF.

* The user should not be allowed to change the primary fields.
* The sheet is protected against change and a particular range will
  be unprotected for editing

* Create a range to enable editing for non key fields
PERFORM f_create_range USING 9 "Begin on 9th row
  4 "Begin on 4th col
  wf_entries "No of rows reqd
  2 "Only 2 columns are editable
  'EDIT'. "Range name

* Unprotect the range for editing
CALL METHOD r_excel->protect_range
  EXPORTING
    name = 'EDIT'
    protect = space
IMPORTING
    error = r_error
    retcode = wf_retcode.

IF r_error->has_failed = abap_true.
  CALL METHOD r_error->raise_message
    EXPORTING
      type = 'E'.
  ENDIF.

*->Set colour to editable range
*# Check SAP help link for colour codes
* http://help.sap.com/saphelp_NW04s/helpdata/en"/21/b531bfe1ba11d2bdbe080009b4534c/frameset.htm
CALL METHOD r_excel->set_color
  EXPORTING
    rangename = 'EDIT'
    front = 1
    back = 4
IMPORTING
    error = r_error
    retcode = wf_retcode.

IF r_error->has_failed = abap_true.
  CALL METHOD r_error->raise_message
    EXPORTING
      type = 'E'.
  ENDIF.

ENDFORM. " f_protect_sheet
*----------------------------------------------------------------------*
* & Form f_close_document
* Close the document when user leaves the program
*---------------------------------------------------------------------*
FORM f_close_document .
* Close document
  IF NOT r_proxy IS INITIAL.
    CALL METHOD r_proxy->close_document
      IMPORTING
        error = r_error
        retcode = wf_retcode.
  IF r_error->has_failed = abap_true.
    CALL METHOD r_error->raise_message
      EXPORTING
        type = 'E'.
  ENDIF.
ENDIF.
ENDFORM. " f_close_document

* Form f_save_document
*---------------------------------------------------------------------*
* Save the modified entries into database table
*---------------------------------------------------------------------*
FORM f_save_document .
DATA: locint_ranges   TYPE soi_range_list,
     locwa_ranges   TYPE soi_range_item,
     locint_moddata  TYPE soi_generic_table,
     locwa_moddata  TYPE soi_generic_item,
     locint_agents_mod  TYPE TABLE OF ysm_agents,
     locwa_agents_mod  TYPE ysm_agents,
     loc_error_row   TYPE i.

* Initialize the colour of the editable range
CALL METHOD r_excel->set_color
  EXPORTING
    rangename = 'EDIT'
    front    = 1
    back     = 4
  IMPORTING
    error   = r_error
    retcode = wf_retcode.

  IF r_error->has_failed = abap_true.
    CALL METHOD r_error->raise_message
      EXPORTING
        type = 'E'.
  ENDIF.

* Define the range from which data needs to be read
locwa_ranges-name   = 'AGENTS'.
locwa_ranges-rows   = wf_entries.
locwa_ranges-columns = 3.
APPEND locwa_ranges TO locint_ranges.

* Get modified data
CALL METHOD r_excel->get_ranges_data
  IMPORTING
    contents = locint_moddata
    error = r_error
  CHANGING
    ranges = locint_ranges.

IF r_error->has_failed = abap_true.
  CALL METHOD r_error->raise_message
    EXPORTING
      type = 'E'.
ENDIF.

LOOP AT locint_moddata INTO locwa_moddata.
  CASE locwa_moddata-column.
    WHEN 1.
      locwa_agents_mod-agentid = locwa_moddata-value.
    WHEN 2.
      locwa_agents_mod-name = locwa_moddata-value.
    WHEN 3.
      locwa_agents_mod-email = locwa_moddata-value.
  END_CASE.
  AT END OF row.
    locwa_agents_mod-mandt = sy-mandt.
    APPEND locwa_agents_mod TO locint_agents_mod.
    CLEAR locwa_agents_mod.
  ENDAT.

ENDLOOP.

* Update Table
  MODIFY ysm_agents FROM TABLE locint_agents_mod.
  COMMIT WORK.

  IF sy-subrc EQ 0.
    MESSAGE 'DATA UPDATED' TYPE 'S'.
    ELSE.
    MESSAGE 'DATA NOT UPDATED' TYPE 'E'.
  ENDIF.

ENDFORM.  " f_save_document

*/--------------------------------------------------------------------*/
*/ Form f_validate_email  */--------------------------------------------------------------------*/
*/ Validate the email id entered  */--------------------------------------------------------------------*/
*/ --> l_email Email Id  */--------------------------------------------------------------------*/
FORM f_validate_email USING  l_email  TYPE c
_err_row TYPE i.

TYPE-POOLS: sx.
DATA: locwa_address TYPE sx_address.

* Check Email Id
 locwa_address-type = 'INT'.
 locwa_address-address = l_email.

CALL FUNCTION 'SX_INTERNET_ADDRESS_TO_NORMAL'
 EXPORTING
   address_unstruct = locwa_address
 EXCEPTIONS
   error_address_type = 1
   error_address = 2
   error_group_address = 3
   OTHERS = 4.

IF sy-subrc <> 0.

* Create a range to highlight the error cell
 PERFORM f_create_range USING _err_row
   5 "Column no for email id
   1
   1 'ERROR'.

* Display the error cell in red
 CALL METHOD r_excel->set_color
 EXPORTING
   rangename = 'ERROR'
   front = 1
   back = 3
 IMPORTING
   error = r_error
   retcode = wf_retcode.

IF r_error->has_failed = abap_true.
 CALL METHOD r_error->raise_message
 EXPORTING
   type = 'E'.
 ENDIF.

MESSAGE 'Invalid Email Address' TYPE 'E'.
ENDIF.

ENDFORM. " f_validate_email
*---------------------------------------------------------------*
* & Form f_create_range
*---------------------------------------------------------------*
* Create a range dynamically in excel sheet
*---------------------------------------------------------------*
* -- > l_top Begin on row
* -- > l_left Begin on column
* -- > l_row No of rows
* -- > l_column No of columns
* -- > l_range Range Name
*---------------------------------------------------------------*
FORM f_create_range USING l_top TYPE i
l_left  TYPE i
l_row   TYPE i
l_column TYPE i
l_range  TYPE char255.

* Select area for entries to be displayed
CALL METHOD r_excel->set_selection
  EXPORTING
    top     = l_top
    left    = l_left
    rows    = l_row
    columns = l_column.

* Define Range
CALL METHOD r_excel->insert_range
  EXPORTING
    name     = l_range
    rows     = l_row
    columns  = l_column
  IMPORTING
    error    = r_error.

IF r_error->has_failed = abap_true.
  CALL METHOD r_error->raise_message
    EXPORTING
      type = 'E'.
ENDIF.

ENDFORM.       " f_create_range

Selection Screen:

<table>
<thead>
<tr>
<th>Agent Id</th>
<th>A123*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Name in OAOR</td>
<td>YSM_TESTBDS</td>
</tr>
<tr>
<td>Class Type in OAOR</td>
<td>OT</td>
</tr>
<tr>
<td>Object key in OAOR</td>
<td>TEST</td>
</tr>
<tr>
<td>Description of Table Template</td>
<td>TABLE TEMPLATE</td>
</tr>
</tbody>
</table>

Initial Screen Displaying the Table Entries:
Doesn’t allow the user to modify the primary field AGENTID or rest of the cells, except the fields NAME & EMAILID:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>AGENTID</td>
<td>NAME OF THE AGENT</td>
<td>EMAIL ID</td>
</tr>
<tr>
<td>9</td>
<td>A12345</td>
<td>User1</td>
<td><a href="mailto:abc@sap.com">abc@sap.com</a></td>
</tr>
<tr>
<td>10</td>
<td>A12346</td>
<td>User2</td>
<td><a href="mailto:abc@sap.com">abc@sap.com</a></td>
</tr>
<tr>
<td>11</td>
<td>A12347</td>
<td>User3</td>
<td><a href="mailto:abc@sap.com">abc@sap.com</a></td>
</tr>
<tr>
<td>12</td>
<td>A12348</td>
<td>User4</td>
<td><a href="mailto:abc@sap.com">abc@sap.com</a></td>
</tr>
<tr>
<td>13</td>
<td>A12349</td>
<td>User5</td>
<td><a href="mailto:abc@sap.com">abc@sap.com</a></td>
</tr>
<tr>
<td>14</td>
<td>A12350</td>
<td>User6</td>
<td><a href="mailto:abc@sap.com">abc@sap.com</a></td>
</tr>
<tr>
<td>15</td>
<td>A12351</td>
<td>User7</td>
<td><a href="mailto:abc@sap.com">abc@sap.com</a></td>
</tr>
<tr>
<td>16</td>
<td>A12352</td>
<td>User8</td>
<td><a href="mailto:abc@sap.com">abc@sap.com</a></td>
</tr>
<tr>
<td>17</td>
<td>A12353</td>
<td>User9</td>
<td><a href="mailto:abc@sap.com">abc@sap.com</a></td>
</tr>
<tr>
<td>18</td>
<td>A12354</td>
<td>User10</td>
<td><a href="mailto:abc@sap.com">abc@sap.com</a></td>
</tr>
<tr>
<td>19</td>
<td>A12355</td>
<td>User11</td>
<td><a href="mailto:abc@sap.com">abc@sap.com</a></td>
</tr>
</tbody>
</table>

Shows error message and highlights the cell to be corrected in red, if an invalid email id is entered:
Data Changed and Successfully Saved:

---

**Data Browser: Table YSM**

<table>
<thead>
<tr>
<th>CI</th>
<th>Agent Id</th>
<th>Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>800</td>
<td>A1 2345</td>
<td>Agent1</td>
<td><a href="mailto:xyz@sap.com">xyz@sap.com</a></td>
</tr>
<tr>
<td>800</td>
<td>A1 2346</td>
<td>Agent2</td>
<td><a href="mailto:xyz@sap.com">xyz@sap.com</a></td>
</tr>
<tr>
<td>800</td>
<td>A1 2347</td>
<td>Agent3</td>
<td><a href="mailto:xyz@sap.com">xyz@sap.com</a></td>
</tr>
<tr>
<td>800</td>
<td>A1 2348</td>
<td>Agent4</td>
<td><a href="mailto:xyz@sap.com">xyz@sap.com</a></td>
</tr>
<tr>
<td>800</td>
<td>A1 2349</td>
<td>Agent5</td>
<td><a href="mailto:xyz@sap.com">xyz@sap.com</a></td>
</tr>
<tr>
<td>800</td>
<td>A1 2350</td>
<td>Agent6</td>
<td><a href="mailto:xyz@sap.com">xyz@sap.com</a></td>
</tr>
<tr>
<td>800</td>
<td>A1 2351</td>
<td>Agent7</td>
<td><a href="mailto:xyz@sap.com">xyz@sap.com</a></td>
</tr>
<tr>
<td>800</td>
<td>A1 2352</td>
<td>Agent8</td>
<td><a href="mailto:xyz@sap.com">xyz@sap.com</a></td>
</tr>
<tr>
<td>800</td>
<td>A1 2353</td>
<td>Agent9</td>
<td><a href="mailto:xyz@sap.com">xyz@sap.com</a></td>
</tr>
<tr>
<td>800</td>
<td>A1 2354</td>
<td>Agent10</td>
<td><a href="mailto:xyz@sap.com">xyz@sap.com</a></td>
</tr>
<tr>
<td>800</td>
<td>A1 2355</td>
<td>Agent11</td>
<td><a href="mailto:xyz@sap.com">xyz@sap.com</a></td>
</tr>
</tbody>
</table>
Event Handler Technique in Object oriented ABAP

Event is a mechanism by which method of one class can raise method of another class, without the hazard of instantiating that class. It provides to raise the method (event handler method) of one class with help of another method in the same or different class (triggering method).

The below steps is required to have the event handler in the class:

- Create an event in a class.
- Create a triggering method in the same class which will raise the event.
- Create an event handler method for the event in same/other class.
- Register the event handler method in the program.

Now, the above settings are complete for event handler in class. Create an object from the class containing the event and call the triggering method to raise the event.

By taking the above steps, the following sample examples will demonstrate the event handler technique in Class.

1. **Events with Handler Method in the same class.**

   This example tells that how to raise method, if the triggering method and event handler method presents in the same class.

   **Sample code and Output.**
REPORT ZCLASS_EVENT001.

CLASS class1 DEFINITION.
  PUBLIC SECTION.
  * Creating event : Event1
  EVENTS: EVENT1.
  * Creating an event handling method. This method can belong to
  * same or different class
  METHODS: METHOD1 FOR EVENT EVENT1 OF CLASS1.
  * Method to raise the event
  METHODS : TRIGGER1.
ENDCLASS.  "class1 DEFINITION

CLASS class1 IMPLEMENTATION.
  *
  CLASS class1 IMPLEMENTATION

  "class1 IMPLEMENTATION

  *
  METHOD : METHOD1.
  WRITE:/5 'I am the event handler method'.
  ENDMETHOD.
  ".
  *
  METHOD : TRIGGER1.
  WRITE:/5 'I am Trigger1, going to raise event Event1'.
  ".
ABAP Editor: Change Report ZCLASS_EVENT001

```
METHOD : TRIGGER1.
  WRITE:/5 'I am Trigger1, going to raise event Event1'.
  RAISE EVENT EVENT1.
ENDMETHOD.
ENDCLASS. "class1 IMPLEMENTATION"

START-OF-SELECTION.
  DATA: OBJ TYPE REF TO CLASS1.
  CREATE OBJECT: OBJ.
  "Registering the event handler method"
  SET HANDLER OBJ->METHOD1 FOR OBJ .
  "Calling the event which will raise the event."
  CALL METHOD OBJ->TRIGGER1.
```

Output.

### Events with Handler Method in the same class

I am Trigger1, going to raise event Event1
I am the event handler method

Now select the method.
And also copy the parameters of the event method.
By pressing this copy event parameter we can get the parameters.

Save and go back to the earlier screen.

Then double click on the method name.

Then provide the following logic for triggering the event.

METHOD METHOD_EVENT.

*check the condition

IF S_LIFNR_LOW < 1000 AND S_LIFNR_HIGH > 2000.
   MESSAGE I000(0) WITH 'enter the values between 1000 and 2000'.
   RAISE EVENT ZEVENT_METHOD.
ENDIF.

*provide select statement

SELECT *
FROM LFA1
INTO TABLE IT_LFA1
WHERE LIFNR BETWEEN S_LIFNR_LOW AND S_LIFNR_HIGH.

*transfer the values to another internal table

IT_LFA11 = IT_LFA1.
ENDMETHOD.
After that provide the logic in se38.
REPORT  ZCL_EVENT_OPERATION .
*provide data objects
DATA: LFA1 TYPE LFA1,
    OBJ TYPE REF TO ZCL_EVENT_OPERATION,
    IT_LFA1 TYPE Z_LFA1,
    IT_LFA11 TYPE Z_LFA1,
    WA_LFA1 TYPE LFA1.
*provide select statement
SELECT OPTIONS: S_LIFNR FOR LFA1-LIFNR.
*provide create object
CREATE OBJECT SELECTION.
*call the method
CALL METHOD OBJ->METHOD_EVENT
    EXPORTING
    S_LIFNR_LOW = S_LIFNR-LOW
    S_LIFNR_HIGH = S_LIFNR-HIGH
    IT_LFA1 = IT_LFA1.
*provide attribute value
IT_LFA11 = OBJ->IT_LFA11.
*display the data
LOOP AT IT_LFA11 INTO WA_LFA1.
  WRITE:/ WA_LFA1-LIFNR,
       WA_LFA1-LAND1,
       WA_LFA1-NAME1,
       WA_LFA1-ORT01.
ENDLOOP.
Save it, check it, activate it and execute it.

Then the output is like this.

If lifnr value is <1000 and >2000.
Then press execute it.

The output is like this.
Then press enter.

The output is like this.
Dialog processing after COMMIT WORK statement

How to perform dialog processing after commit work execution?

In general, we may come across the scenario where, some dialog processing needs to be done after transaction “commit work”. It’s explained here by considering a scenario.

After filling all necessary details in the delivery document, user clicks on “save” button to create a delivery document. If any dialog processing (like pop-up to fill some details) required upon successful execution of COMMIT WORK statement. In this case, we can approach below method.

Let me explain this by creating a custom class.

Create an event handler method in the custom class ZTEST_HANDLER for the event TRANSACTION_FINISHED of the standard class CL_SYSTEM_TRANSACTION_STATE.

Standard class: CL_SYSTEM_TRANSACTION_STATE

Event name : TRANSACTION_FINISHED

Note: This event gets triggered as soon as the COMMIT WORK gets executed.

My custom class name : ZTEST_HANDLER

My event handler method: CALL_DIALOG (Event TRANSACTION_FINISHED of standard class CL_SYSTEM_TRANSACTION_STATE attached to this custom method)

1) Event handler method CALL_DIALOG

2) Event handler method: CALL_DIALOG detailed view
Once the COMMIT WORK for the transaction is executed, control comes to custom method CALL_DIALOG method. Here we can check whether transaction got committed successfully or rolled back by using interface parameter KIND as shown in below screen shot.

To get the control to the CALL_DIALOG method, we need to do SET HANDLER to register the event in any user exit before transaction COMMIT WORK execution.

Here in this case, I registered event in a BADI, which gets triggered after pressing SAVE button in the outbound delivery (VL01N/VL02N) and before COMMIT WORK execution.

Please find below screen shot of BADI method.
The Event TRANANSACTION_FINISHED of standard Class CL_SYSTEM_TRANSACTION_STATE and its parameters are shown in below screen shots:

Attributes of Class CL_SYSTEM_TRANSACTION_STATE:
Note: We can use IMPORT and EXPORT statements to transfer data from BADI to the method CALL_DIALOG.