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>Procedural 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
2. Implementation

**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 ... ENDCCLASS.

```abap
CLASS <class> DEFINITION. . ENDCCLASS.
```

**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. . ENDCCLASS.
```

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.

```abap
REPORT YDEMO_CLASS.
  *  CLASS class1 DEFINITION
  *
  CLASS class1 DEFINITION.
  PUBLIC SECTION.
  DATA:
  w_text(40) VALUE 'ABAP Objects'.
  METHODS: DISPLAY.
  ENDCLASS. "Parentcl DEFINITION
  &
  Class (Implementation) class1
  &
  Text
  CLASS class1 IMPLEMENTATION.
  METHOD DISPLAY.
  WRITE: / 'This is method 'DISPLAY''.
  ENDMETHOD. "display
  ENDCLASS. "class1

START-OF-SELECTION.

  DATA:
  class1 TYPE REF TO class1.
  CREATE OBJECT: class1.
  WRITE: / class1->w_text.
  CALL METHOD:
  class1->DISPLAY.
```

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 Inheritance: Acquiring the properties from a single parent. (Children can be more).

![Diagram of Single Inheritance](image)

Example for Single Inheritance

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

Example
Syntax: \texttt{CLASS <subclass> DEFINITION INHERITING FROM <superclass>}. 

Let us see a very simple example for creating subclass(child) from a superclass(parent)

```plaintext
REPORT YDEMO_INHERITANCE.

CLASS PARENT DEFINITION

CLASS PARENT DEFINITION.
  PUBLIC SECTION.
    DATA:
      W_PUBLIC(30) VALUE 'This is public data'.
    METHODS: PARENTMET.
  ENDCLASS.  "Parentcl DEFINITION

CLASS Child DEFINITION INHERITING FROM PARENT.
  PUBLIC SECTION.
    METHODS: CHILDMET.
  ENDCLASS.  "child DEFINITION INH

CLASS PARENT IMPLEMENTATION.
  METHOD PARENTMET.
    WRITE /: W_PUBLIC.
  ENDMETHOD.  "Display

ENDCLASS.  "PARENT

CLASS Child IMPLEMENTATION.
  METHOD CHILDMET.
    WRITE /: W_PUBLIC.
  ENDMETHOD.  "Display

ENDCLASS.  "CHILDMET
```

*&&&

```
Text
```
Multiple Inheritance is **not supported** by ABAP.

Output is as follows:

<table>
<thead>
<tr>
<th>Test program for Inheritance</th>
</tr>
</thead>
<tbody>
<tr>
<td>This is public data</td>
</tr>
<tr>
<td>Method in child class</td>
</tr>
<tr>
<td>This is public data</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Level</th>
<th>Visibility</th>
<th>Modeled</th>
<th>Method type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPOSIT</td>
<td>Instance</td>
<td>_Public</td>
<td></td>
<td></td>
<td>Adds a deposit</td>
</tr>
<tr>
<td>SET_BALANCE</td>
<td>Instance</td>
<td>_Public</td>
<td></td>
<td></td>
<td>Sets the balance</td>
</tr>
<tr>
<td>WITHDRAW</td>
<td>Instance</td>
<td>_Public</td>
<td></td>
<td></td>
<td>Subtracts</td>
</tr>
</tbody>
</table>

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.
Write the required code. Similarly for SET_BALANCE.

```
<table>
<thead>
<tr>
<th>Method</th>
<th>SET_BALANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```

Similarly for WITHDRAW.

```
<table>
<thead>
<tr>
<th>Method</th>
<th>WITHDRAW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```

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.
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

Lets 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

<table>
<thead>
<tr>
<th>Action</th>
<th>Amount</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set balance</td>
<td>4,000.00</td>
<td>to</td>
</tr>
<tr>
<td>Deposited</td>
<td>3,000.00</td>
<td>bucks making balance to</td>
</tr>
<tr>
<td>Withdrew</td>
<td>200.00</td>
<td>bucks making balance to</td>
</tr>
</tbody>
</table>
**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.
**Working with events in a Global Class**

Go to Class Builder “SE24”.

Provide class name.

![Class Builder: initial Screen](image)

Press create button.

![Create Class ZCL_EVENT_OPERATION](image)

Save it.

Go to event tab.

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Optional</th>
<th>Type</th>
<th>Associated Type</th>
<th>Defaultvalue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5_LIFMR_LOW</td>
<td></td>
<td>5_LIFMR</td>
<td>Account Number of Vendor or Creditor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5_LIFMR_HIGH</td>
<td></td>
<td>5_LIFMR</td>
<td>Account Number of Vendor or Creditor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11_LFA1</td>
<td></td>
<td>5_LFA1</td>
<td>Vendor Master (General Section)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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_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.
Click4interview

Debug yourself ...
Paste the Below Code.

```abap
*&---------------------------------------------------------------*
*& 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:

Execute the program:
```

![Testing Enhancement](https://www.saptechnical.com)