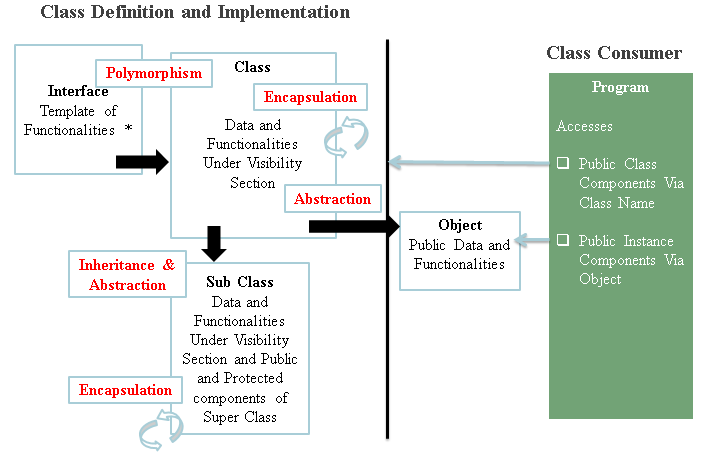
Features of OOPs

What are the advantages/Features of OOPs?

* Multiple copies or Multiple Instances
* Encapsulation
* Abstraction
* Inheritance
* Events
* Polymorphism



**Recommended:**

Basic knowledge on syntaxes of how to define and implement the class and calling methods of the class

Contents

[Multiple Instances/Copies 3](#_Toc533948848)

[Encapsulation: Binding Logic with Data 4](#_Toc533948849)

[Program: 5](#_Toc533948850)

[Abstraction 9](#_Toc533948851)

[Inheritance 11](#_Toc533948852)

[Adding New Functionality 11](#_Toc533948853)

[Program 12](#_Toc533948854)

[Redefine Existing Functionality 14](#_Toc533948855)

[Program 16](#_Toc533948856)

[Events 19](#_Toc533948857)

[Program 20](#_Toc533948858)

[Class: Customer 20](#_Toc533948859)

[Class: Lazy Department 20](#_Toc533948860)

[Class: Active Department 21](#_Toc533948861)

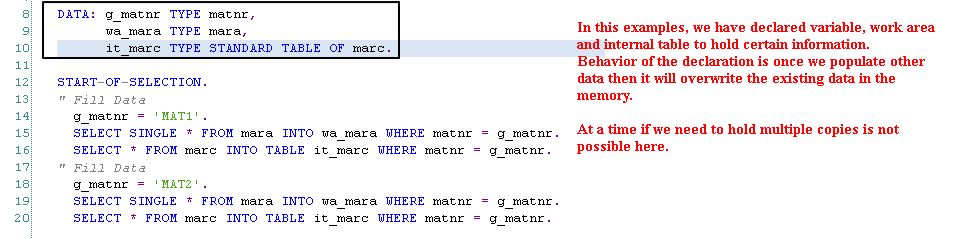
[Polymorphism 23](#_Toc533948862)

[Program 23](#_Toc533948863)

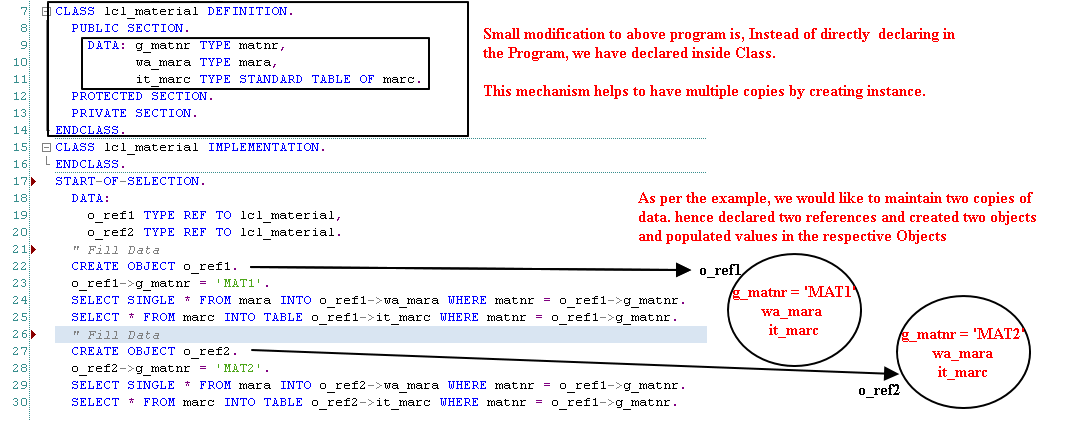
# Multiple Instances/Copies

When it is needed to maintain multiple copies of same field then we will bind the data in the class, create instance and will use

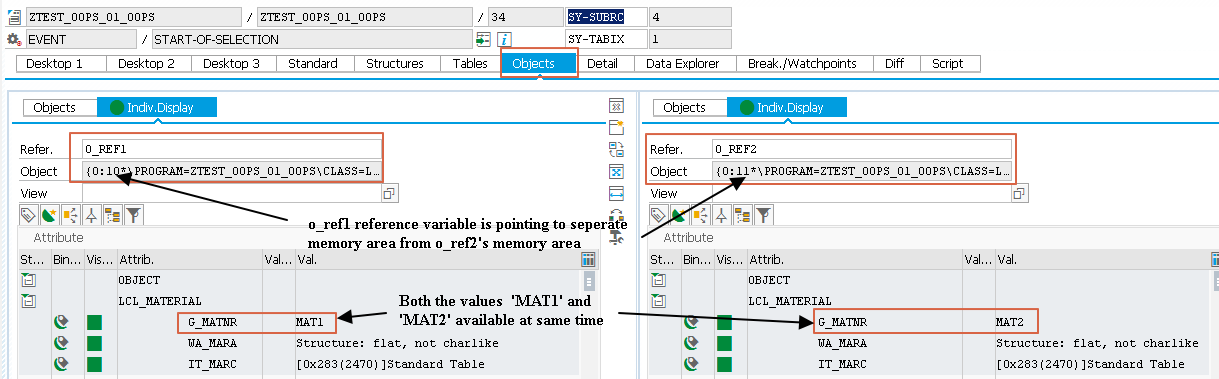
**Program with Without Object Orientation**



**Program with Object Oriented Features**



In the debugging, we can observe that memory is live for both the instances.



# Encapsulation: Binding Logic with Data

Encapsulation allows to bind the logic with related data only.

Let assume that, application program is going to deal with Customer Master and Sales Order details and corresponding operations

**Data**: Customer Master and Sales Order Data

**Operations**: Create Customer Master, Change Customer Master, Display Customer Master, Block Customer, Release Customer, Create Sales Order, Change Sales Order, Display Sales Order, Deliver Sales Order, Bill Sales Order

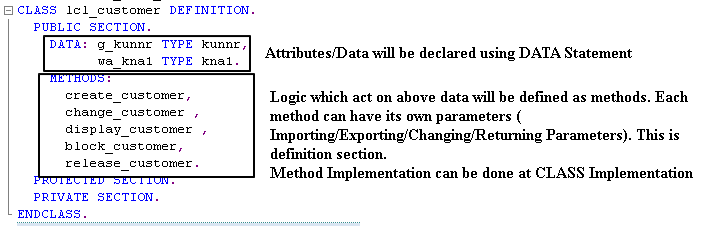
In Procedure Oriented Programming, we will define all the data (There is not separation) in one location (Ideally TOP Include in the ABAP Program) and all the operations defined inform of routines one by one.

In the Object Orientation, We can separate the data and corresponding logic and group them in separate classes

**Class#1: Customer Class**

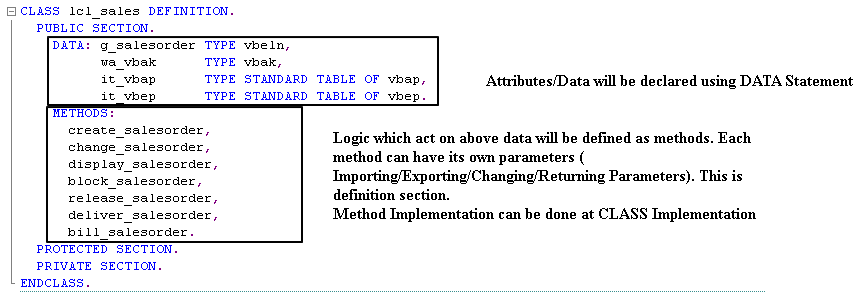
Data: Customer Master Data

Operations: Create Customer, Change Customer, Display Customer, Block Customer, Release Customer



**Class#2: Sales Class**

Operations: Create Sales Order, Changes Sales Order, Display Sales Order, Deliver Sales Order, Bill Sales Order



As per the above example, based on the data and corresponding operation, we are able to separate the data and logic into two different units.

*Note: Below example is used for Abstraction example also as we have used on Method Next\_Action in the protected section.*

## Program:

\*&---------------------------------------------------------------------\*  
\*& Report ZTEST\_OOPS\_02  
\*&---------------------------------------------------------------------\*  
\*&  
\*&---------------------------------------------------------------------\*  
REPORT ztest\_oops\_02.  
CLASS lcl\_customer DEFINITION.  
  PUBLIC SECTION.  
    DATA:  
      g\_kunnr TYPE kunnr,  
      wa\_kna1 TYPE kna1.  
    METHODS:  
      create\_customer,  
      change\_customer,  
      display\_customer,  
      block\_customer,  
      release\_customer.  
  PROTECTED SECTION.  
    METHODS next\_action.  
  PRIVATE SECTION.  
    METHODS:  
      send\_cust\_sts\_to\_recovery\_team,  
      send\_cust\_sts\_to\_legal\_team,  
      send\_cust\_sts\_to\_sales\_team,  
      send\_cust\_sts\_to\_local\_rowdies.  
ENDCLASS.  
  
CLASS lcl\_sales DEFINITION.  
  PUBLIC SECTION.  
    DATA:  
      g\_salesorder TYPE vbeln,  
      wa\_vbak      TYPE vbak,  
      it\_vbap      TYPE STANDARD TABLE OF vbap,  
      it\_vbep      TYPE STANDARD TABLE OF vbep.  
    METHODS:  
      create\_salesorder,  
      change\_salesorder,  
      display\_salesorder,  
      block\_salesorder,  
      release\_salesorder,  
      deliver\_salesorder,  
      bill\_salesorder.  
  PROTECTED SECTION.  
    METHODS next\_action.  
  PRIVATE SECTION.  
    METHODS:  
      inform\_ord\_sts\_to\_warehse\_dept,  
      inform\_ord\_sts\_to\_supervisor,  
      inform\_ord\_sts\_to\_market\_team.  
ENDCLASS.  
  
CLASS lcl\_customer IMPLEMENTATION.  
  METHOD create\_customer.  
    WRITE / 'Sales order Created'.  
  ENDMETHOD.  
  METHOD change\_customer.  
    WRITE / 'Sales Order Changed'.  
  ENDMETHOD.  
  METHOD display\_customer.  
    WRITE / 'Display Sales Order'.  
  ENDMETHOD.  
  METHOD next\_action.  
    CALL METHOD:  
      send\_cust\_sts\_to\_recovery\_team,  
      send\_cust\_sts\_to\_legal\_team,  
      send\_cust\_sts\_to\_sales\_team.  
  ENDMETHOD.  
  METHOD block\_customer.  
    WRITE / 'Block Sales Order'.  
    CALL METHOD next\_action.  
  ENDMETHOD.  
  METHOD release\_customer.  
    WRITE / 'Release Customer'.  
  ENDMETHOD.  
  METHOD send\_cust\_sts\_to\_recovery\_team.  
    WRITE / 'Will inform local rowdiesys'.  
  ENDMETHOD.  
  METHOD send\_cust\_sts\_to\_legal\_team.  
    WRITE / 'Proceeding legally takes long time. Customer can fly other countries'.  
    WRITE / 'Better take help of local solution'.  
    WRITE / 'I have amicable solution'.  
  ENDMETHOD.  
  METHOD send\_cust\_sts\_to\_sales\_team.  
    WRITE / 'No more sales.. its fixed..Contact legal team for further action'.  
  ENDMETHOD.  
  METHOD send\_cust\_sts\_to\_local\_rowdies.  
    WRITE / 'Tomorrow morning deadly warning. Recovery money 50-50'.  
  ENDMETHOD.  
ENDCLASS.  
  
CLASS lcl\_sales IMPLEMENTATION.  
  METHOD create\_salesorder.  
    WRITE / 'Sales Order Created'.  
  ENDMETHOD.  
  METHOD change\_salesorder.  
    WRITE / 'Sales Order chaged'.  
  ENDMETHOD.  
  METHOD display\_salesorder.  
    WRITE / 'Sales Order displayed'.  
  ENDMETHOD.  
  METHOD next\_action.  
    CALL METHOD inform\_ord\_sts\_to\_warehse\_dept.  
    CALL METHOD inform\_ord\_sts\_to\_supervisor.  
    CALL METHOD inform\_ord\_sts\_to\_market\_team.  
  ENDMETHOD.  
  METHOD block\_salesorder.  
    WRITE / 'Sales Order Blocked'.  
    CALL METHOD next\_action.  
  ENDMETHOD.  
  METHOD release\_salesorder.  
    WRITE / 'Sales Order released'.  
  ENDMETHOD.  
  METHOD deliver\_salesorder.  
    WRITE / 'Sales Order Delivered'.  
  ENDMETHOD.  
  METHOD bill\_salesorder.  
    WRITE / 'Billing done'.  
  ENDMETHOD.  
  METHOD  inform\_ord\_sts\_to\_warehse\_dept.  
    WRITE /  'Got information from Sales team for fuurther action'.  
  ENDMETHOD.  
  METHOD inform\_ord\_sts\_to\_supervisor.  
    WRITE / 'Got information from Sales team for fuurther action'.  
  ENDMETHOD.  
  METHOD inform\_ord\_sts\_to\_market\_team.  
    WRITE / 'Got information from Sales team for fuurther action'.  
    WRITE / 'As usal we will nothing. that matter sales team dont know'.  
  ENDMETHOD.  
ENDCLASS.  
  
  
START-OF-SELECTION.  
  " Work with Customer  
  DATA:  
     o\_customer TYPE REF TO lcl\_customer.  
  CREATE OBJECT o\_customer.  
  CALL METHOD:  
  o\_customer->create\_customer,  
  o\_customer->change\_customer,  
  o\_customer->display\_customer,  
  o\_customer->block\_customer,  
  o\_customer->release\_customer.  
  
  " Work with Sales Order  
  DATA:  
    o\_salesorder TYPE REF TO lcl\_sales.  
  CREATE OBJECT o\_salesorder.  
  CALL METHOD:  
      o\_salesorder->create\_salesorder,  
      o\_salesorder->change\_salesorder,  
      o\_salesorder->display\_salesorder,  
      o\_salesorder->block\_salesorder,  
      o\_salesorder->release\_salesorder,  
      o\_salesorder->deliver\_salesorder,  
      o\_salesorder->bill\_salesorder.

Output:



# Abstraction

Abstraction allows controlling access to class features i.e. on Data and Methods. Class contains 3 visibility sections.

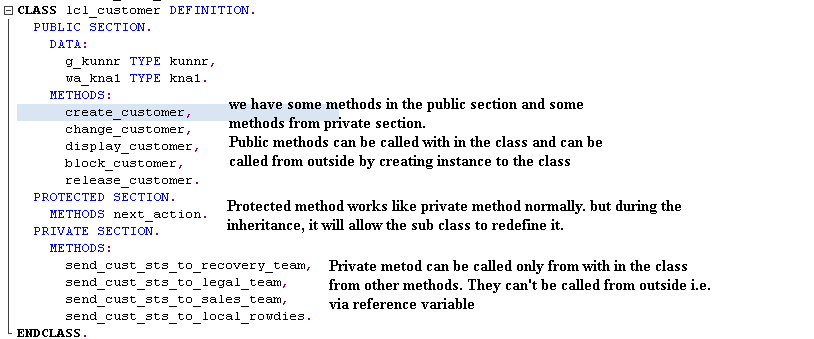
* Public Section
* Protected Section
* Private Section

**Public Section:** Whatever the components (Data or Methods) declared in this section can be accessed with in the Class as well as outside the class (From the consumer) by declaring the reference and creating instance

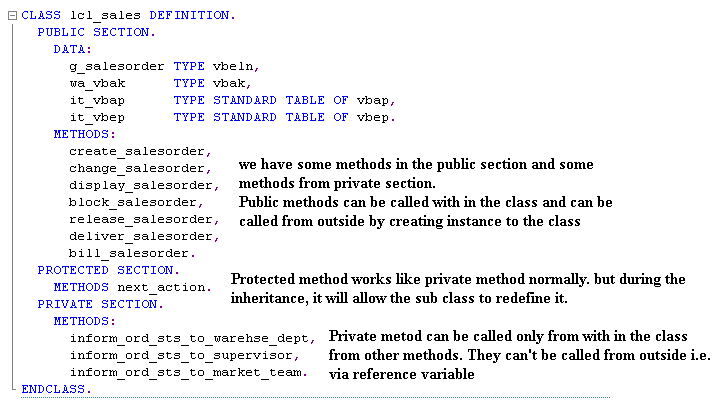
**Protected Section:** This section basically for extending features of the existing class. Whatever the class components which can be extended/re-implemented in the future can be specified

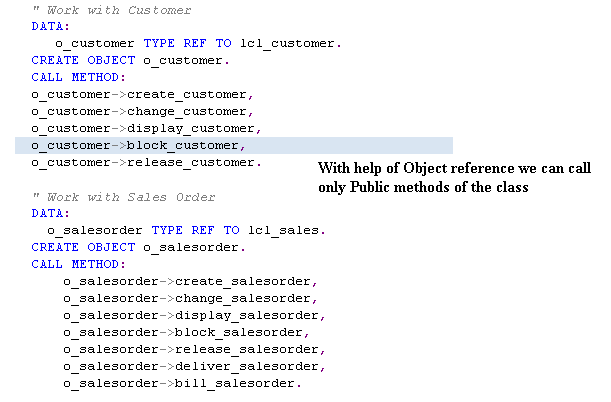
**Private Section**: this section contains only those components which can be used only with in the class

**Customer Class:**



**Sales Class:**





# Inheritance

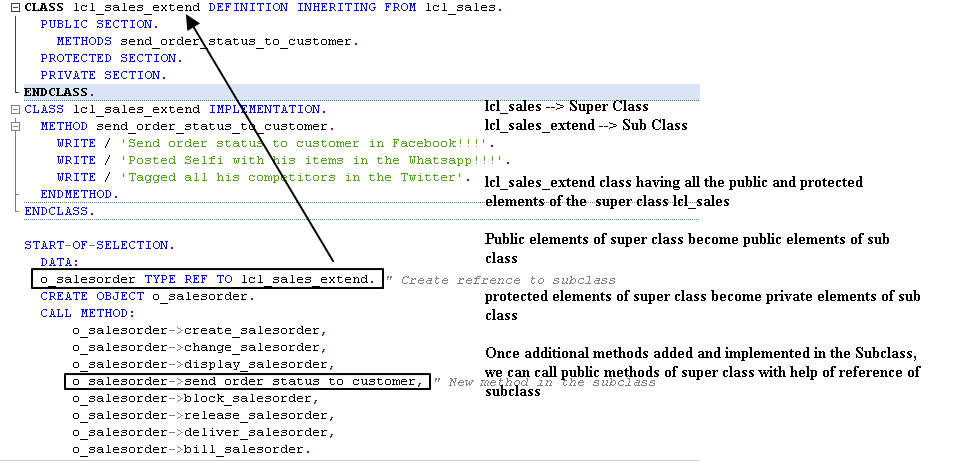
Inheritance allows to extend the existing functionality of class.

We can

* Add new elements data, events and methods in the sub class
* Redefine existing Functionality

## Adding New Functionality

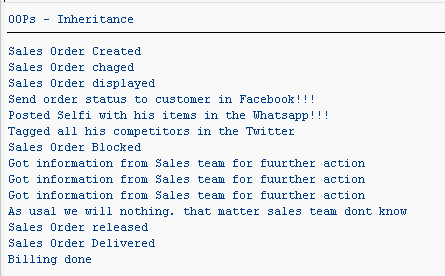
Add new Functionality **“Send\_Order\_status\_to\_the\_Customer”** to the Sales Class by extending the class



### Program

CLASS lcl\_sales DEFINITION.  
  PUBLIC SECTION.  
    DATA:  
      g\_salesorder TYPE vbeln,  
      wa\_vbak      TYPE vbak,  
      it\_vbap      TYPE STANDARD TABLE OF vbap,  
      it\_vbep      TYPE STANDARD TABLE OF vbep.  
    METHODS:  
      create\_salesorder,  
      change\_salesorder,  
      display\_salesorder,  
      block\_salesorder,  
      release\_salesorder,  
      deliver\_salesorder,  
      bill\_salesorder.  
  PROTECTED SECTION.  
    METHODS next\_action.  
  PRIVATE SECTION.  
    METHODS:  
      inform\_ord\_sts\_to\_warehse\_dept,  
      inform\_ord\_sts\_to\_supervisor,  
      inform\_ord\_sts\_to\_market\_team.  
ENDCLASS.  
  
CLASS lcl\_sales IMPLEMENTATION.  
  METHOD create\_salesorder.  
    WRITE / 'Sales Order Created'.  
  ENDMETHOD.  
  METHOD change\_salesorder.  
    WRITE / 'Sales Order chaged'.  
  ENDMETHOD.  
  METHOD display\_salesorder.  
    WRITE / 'Sales Order displayed'.  
  ENDMETHOD.  
  METHOD next\_action.  
    CALL METHOD inform\_ord\_sts\_to\_warehse\_dept.  
    CALL METHOD inform\_ord\_sts\_to\_supervisor.  
    CALL METHOD inform\_ord\_sts\_to\_market\_team.  
  ENDMETHOD.  
  METHOD block\_salesorder.  
    WRITE / 'Sales Order Blocked'.  
    CALL METHOD next\_action.  
  ENDMETHOD.  
  METHOD release\_salesorder.  
    WRITE / 'Sales Order released'.  
  ENDMETHOD.  
  METHOD deliver\_salesorder.  
    WRITE / 'Sales Order Delivered'.  
  ENDMETHOD.  
  METHOD bill\_salesorder.  
    WRITE / 'Billing done'.  
  ENDMETHOD.  
  METHOD  inform\_ord\_sts\_to\_warehse\_dept.  
    WRITE /  'Got information from Sales team for fuurther action'.  
  ENDMETHOD.  
  METHOD inform\_ord\_sts\_to\_supervisor.  
    WRITE / 'Got information from Sales team for fuurther action'.  
  ENDMETHOD.  
  METHOD inform\_ord\_sts\_to\_market\_team.  
    WRITE / 'Got information from Sales team for fuurther action'.  
    WRITE / 'As usal we will nothing. that matter sales team dont know'.  
  ENDMETHOD.  
ENDCLASS.  
  
CLASS lcl\_sales\_extend DEFINITION INHERITING FROM lcl\_sales.  
  PUBLIC SECTION.  
    METHODS send\_order\_status\_to\_customer.  
  PROTECTED SECTION.  
  PRIVATE SECTION.  
ENDCLASS.  
CLASS lcl\_sales\_extend IMPLEMENTATION.  
  METHOD send\_order\_status\_to\_customer.  
    WRITE / 'Send order status to customer in Facebook!!!'.  
    WRITE / 'Posted Selfi with his items in the Whatsapp!!!'.  
    WRITE / 'Tagged all his competitors in the Twitter'.  
  ENDMETHOD.  
ENDCLASS.  
  
START-OF-SELECTION.  
  DATA:  
  o\_salesorder TYPE REF TO lcl\_sales\_extend. " Create refrence to subclass  
  CREATE OBJECT o\_salesorder.  
  CALL METHOD:  
      o\_salesorder->create\_salesorder,  
      o\_salesorder->change\_salesorder,  
      o\_salesorder->display\_salesorder,  
      o\_salesorder->send\_order\_status\_to\_customer, " New method in the subclass  
      o\_salesorder->block\_salesorder,  
      o\_salesorder->release\_salesorder,  
      o\_salesorder->deliver\_salesorder,  
      o\_salesorder->bill\_salesorder.

Output

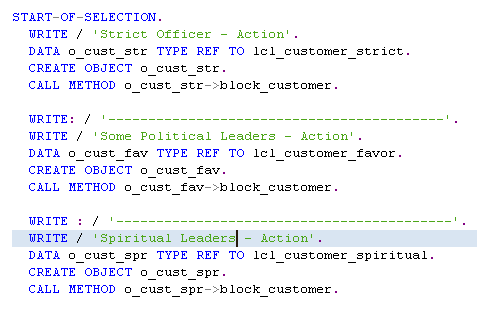


## Redefine Existing Functionality

Redefine allows subclass to have its own implementation instead of Super class implementation



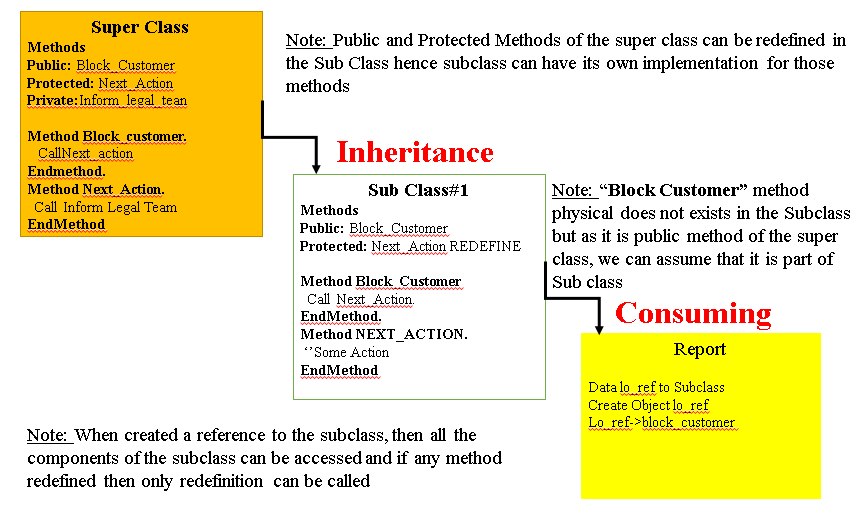
**Creating instances and Calling Block Customer Statement**



In the all the above cases, we have not called the NEXT\_ACTION Method directly. We have called only BLOCK\_CUSTOMER.

BLOCK\_METHOD calling the NEXT\_ACTION Method.

System giving priority to Implementation of Subclass instead of Super class when Method redefined.

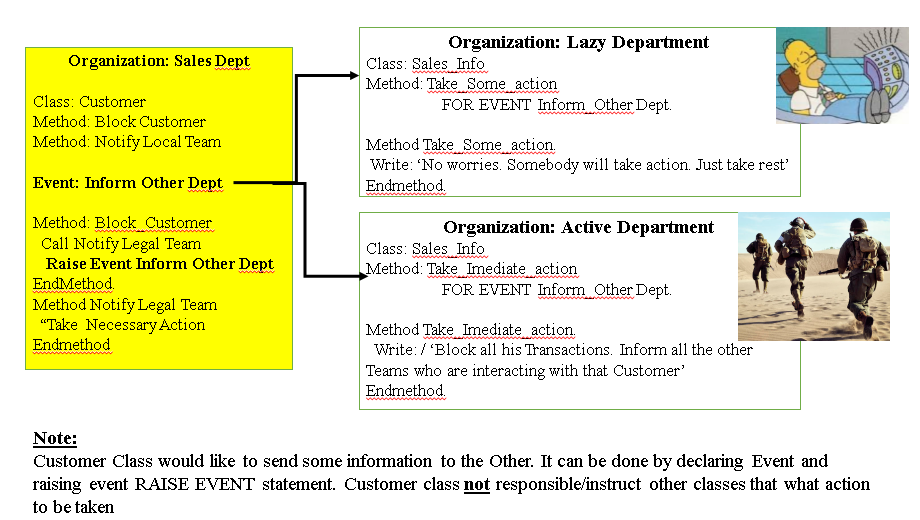


### Program

\*&---------------------------------------------------------------------\*  
\*& Report ZTEST\_OOPS\_04  
\*&---------------------------------------------------------------------\*  
\*&  
\*&---------------------------------------------------------------------\*  
REPORT ztest\_oops\_04.  
CLASS lcl\_customer DEFINITION.  
  PUBLIC SECTION.  
    DATA:  
      g\_kunnr TYPE kunnr,  
      wa\_kna1 TYPE kna1.  
    METHODS:  
      create\_customer,  
      change\_customer,  
      display\_customer,  
      block\_customer,  
      release\_customer.  
  PROTECTED SECTION.  
    METHODS next\_action.  
  PRIVATE SECTION.  
    METHODS:  
      send\_cust\_sts\_to\_recovery\_team,  
      send\_cust\_sts\_to\_legal\_team,  
      send\_cust\_sts\_to\_sales\_team,  
      send\_cust\_sts\_to\_local\_rowdies.  
ENDCLASS.  
CLASS lcl\_customer IMPLEMENTATION.  
  METHOD create\_customer.  
    WRITE / 'Sales order Created'.  
  ENDMETHOD.  
  METHOD change\_customer.  
    WRITE / 'Sales Order Changed'.  
  ENDMETHOD.  
  METHOD display\_customer.  
    WRITE / 'Display Sales Order'.  
  ENDMETHOD.  
  METHOD next\_action.  
    CALL METHOD:  
      send\_cust\_sts\_to\_recovery\_team,  
      send\_cust\_sts\_to\_legal\_team,  
      send\_cust\_sts\_to\_sales\_team.  
  ENDMETHOD.  
  METHOD block\_customer.  
    WRITE / 'Block Sales Order'.  
    CALL METHOD next\_action.  
  ENDMETHOD.  
  METHOD release\_customer.  
    WRITE / 'Release Customer'.  
  ENDMETHOD.  
  METHOD send\_cust\_sts\_to\_recovery\_team.  
    WRITE / 'Will inform local rowdiesys'.  
  ENDMETHOD.  
  METHOD send\_cust\_sts\_to\_legal\_team.  
    WRITE / 'Proceeding legally takes long time. Customer can fly other countries'.  
    WRITE / 'Better take help of local solution'.  
    WRITE / 'I have amicable solution'.  
  ENDMETHOD.  
  METHOD send\_cust\_sts\_to\_sales\_team.  
    WRITE / 'No more sales.. its fixed..Contact legal team for further action'.  
  ENDMETHOD.  
  METHOD send\_cust\_sts\_to\_local\_rowdies.  
    WRITE / 'Tomorrow morning deadly warning. Recovery money 50-50'.  
  ENDMETHOD.  
ENDCLASS.  
  
CLASS lcl\_customer\_strict DEFINITION  
  INHERITING FROM lcl\_customer.  
  PUBLIC SECTION.  
  PROTECTED SECTION.  
    METHODS next\_action REDEFINITION.  
  PRIVATE SECTION.  
ENDCLASS.  
CLASS lcl\_customer\_strict IMPLEMENTATION.  
  METHOD next\_action.  
    WRITE: / 'Beat him. kick him.'.  
    WRITE: / 'Dont tolerate him...'.  
    WRITE: / ' Agni Agni Agni... ah..ah '.  
  ENDMETHOD.  
ENDCLASS.  
  
CLASS lcl\_customer\_favor DEFINITION  
  INHERITING FROM lcl\_customer.  
  PUBLIC SECTION.  
  PROTECTED SECTION.  
    METHODS next\_action REDEFINITION.  
  PRIVATE SECTION.  
ENDCLASS.  
CLASS lcl\_customer\_favor IMPLEMENTATION.  
  METHOD next\_action.  
    WRITE: / 'Provide all facilities in India'.  
    WRITE: / 'Allow the customer to fly abroad'.  
    WRITE: / 'Manage all the Airport Officials'.  
  ENDMETHOD.  
ENDCLASS.  
CLASS lcl\_customer\_spiritual DEFINITION  
  INHERITING FROM lcl\_customer.  
  PUBLIC SECTION.  
  PROTECTED SECTION.  
    METHODS next\_action REDEFINITION.  
  PRIVATE SECTION.  
ENDCLASS.  
CLASS lcl\_customer\_spiritual IMPLEMENTATION.  
  METHOD next\_action.  
    WRITE: / 'Nothing is permanent. '.  
    WRITE: / 'Why to hurry..Dont worry '.  
    WRITE: / 'Gods observing everything.'.  
    WRITE: / 'Nobody can escape from god'.  
  ENDMETHOD.  
ENDCLASS.  
  
START-OF-SELECTION.  
  WRITE / 'Strict Officer - Action'.  
  DATA o\_cust\_str TYPE REF TO lcl\_customer\_strict.  
  CREATE OBJECT o\_cust\_str.  
  CALL METHOD o\_cust\_str->block\_customer.  
  
  WRITE: / '------------------------------------------'.  
  WRITE / 'Some Political Leaders - Action'.  
  DATA o\_cust\_fav TYPE REF TO lcl\_customer\_favor.  
  CREATE OBJECT o\_cust\_fav.  
  CALL METHOD o\_cust\_fav->block\_customer.  
  
  WRITE : / '------------------------------------------'.  
  WRITE / 'Spiritual Leaders - Action'.  
  DATA o\_cust\_spr TYPE REF TO lcl\_customer\_spiritual.  
  CREATE OBJECT o\_cust\_spr.  
  CALL METHOD o\_cust\_spr->block\_customer.

# Events

Events allows to cast current status to outside. Other classes can use the information to perform certain action.

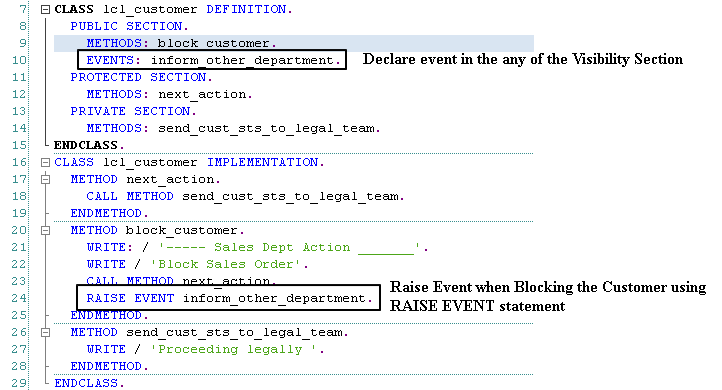


For implementing the Events we have certain steps

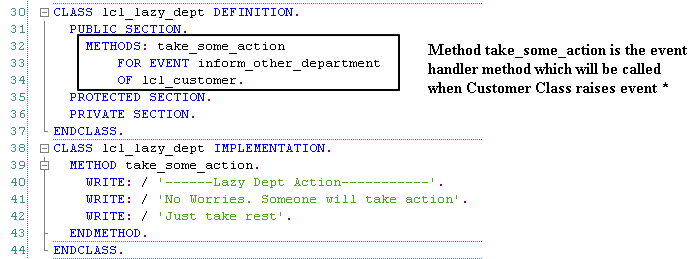
1. Declare Event in the Class 1
2. Raise Event in the Class 2
3. Implement Event Handler Method in Class 2
4. Link the Event Handler method of the Class 2 with Object of Class 1 while Consuming the Class1

## Program

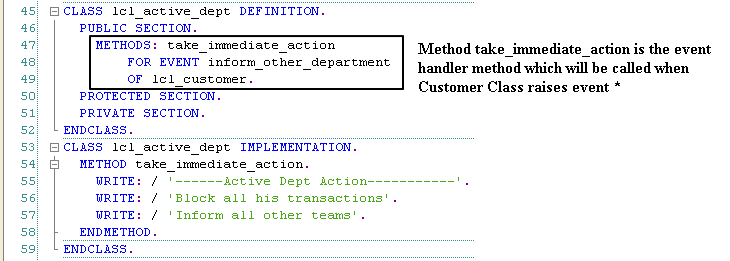
### Class: Customer

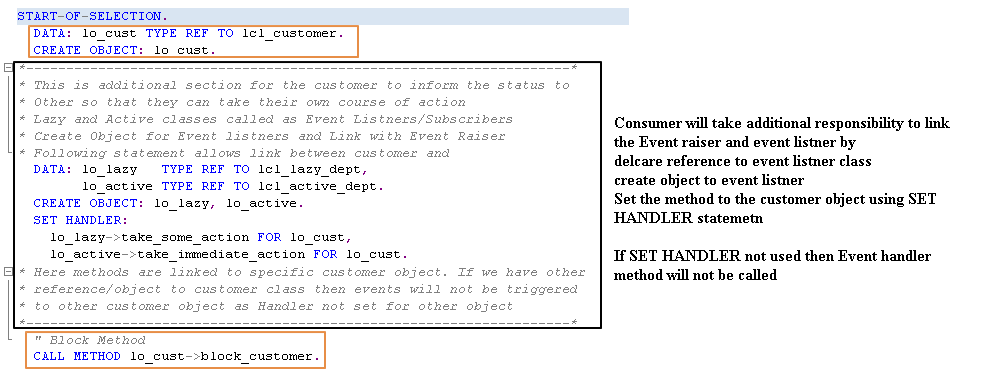


### Class: Lazy Department



### Class: Active Department

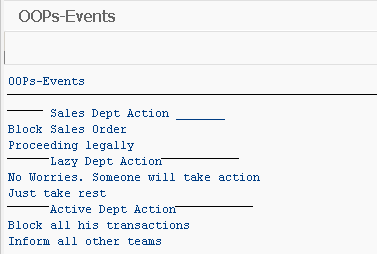




#### Complete Program

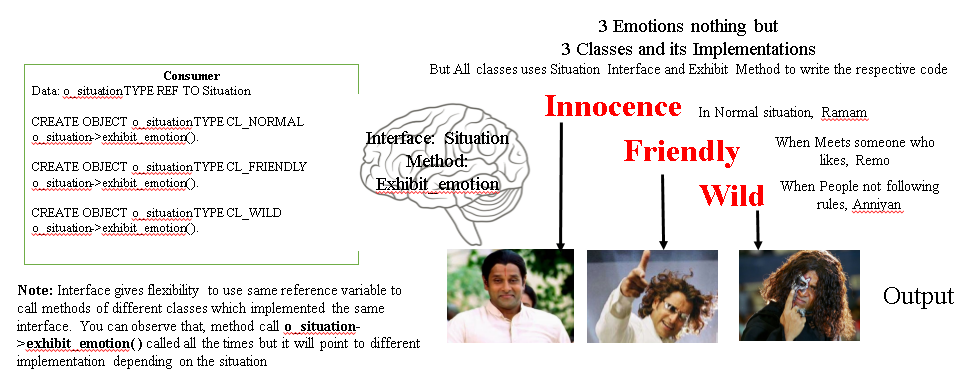
\*&---------------------------------------------------------------------\*  
\*& Report ZTEST\_OOPS\_05  
\*&---------------------------------------------------------------------\*  
\*&  
\*&---------------------------------------------------------------------\*  
REPORT ztest\_oops\_05.  
CLASS lcl\_customer DEFINITION.  
  PUBLIC SECTION.  
    METHODS: block\_customer.  
    EVENTS: inform\_other\_department.  
  PROTECTED SECTION.  
    METHODS: next\_action.  
  PRIVATE SECTION.  
    METHODS: send\_cust\_sts\_to\_legal\_team.  
ENDCLASS.  
CLASS lcl\_customer IMPLEMENTATION.  
  METHOD next\_action.  
    CALL METHOD send\_cust\_sts\_to\_legal\_team.  
  ENDMETHOD.  
  METHOD block\_customer.  
    WRITE: / '----- Sales Dept Action \_\_\_\_\_\_\_'.  
    WRITE / 'Block Sales Order'.  
    CALL METHOD next\_action.  
    RAISE EVENT inform\_other\_department.  
  ENDMETHOD.  
  METHOD send\_cust\_sts\_to\_legal\_team.  
    WRITE / 'Proceeding legally '.  
  ENDMETHOD.  
ENDCLASS.  
CLASS lcl\_lazy\_dept DEFINITION.  
  PUBLIC SECTION.  
    METHODS: take\_some\_action  
        FOR EVENT inform\_other\_department  
        OF lcl\_customer.  
  PROTECTED SECTION.  
  PRIVATE SECTION.  
ENDCLASS.  
CLASS lcl\_lazy\_dept IMPLEMENTATION.  
  METHOD take\_some\_action.  
    WRITE: / '------Lazy Dept Action-----------'.  
    WRITE: / 'No Worries. Someone will take action'.  
    WRITE: / 'Just take rest'.  
  ENDMETHOD.  
ENDCLASS.  
CLASS lcl\_active\_dept DEFINITION.  
  PUBLIC SECTION.  
    METHODS: take\_immediate\_action  
        FOR EVENT inform\_other\_department  
        OF lcl\_customer.  
  PROTECTED SECTION.  
  PRIVATE SECTION.  
ENDCLASS.  
CLASS lcl\_active\_dept IMPLEMENTATION.  
  METHOD take\_immediate\_action.  
    WRITE: / '------Active Dept Action-----------'.  
    WRITE: / 'Block all his transactions'.  
    WRITE: / 'Inform all other teams'.  
  ENDMETHOD.  
ENDCLASS.

START-OF-SELECTION.  
  DATA: lo\_cust TYPE REF TO lcl\_customer.  
  CREATE OBJECT: lo\_cust.  
\*--------------------------------------------------------------------\*  
\* This is additional section for the customer to inform the status to  
\* Other so that they can take their own course of action  
\* Lazy and Active classes called as Event Listners/Subscribers  
\* Create Object for Event listners and Link with Event Raiser  
\*   Following statement allows link between customer and  
  DATA: lo\_lazy   TYPE REF TO lcl\_lazy\_dept,  
        lo\_active TYPE REF TO lcl\_active\_dept.  
  CREATE OBJECT: lo\_lazy, lo\_active.  
  SET HANDLER:  
    lo\_lazy->take\_some\_action FOR lo\_cust,  
    lo\_active->take\_immediate\_action FOR lo\_cust.  
\*--------------------------------------------------------------------\*  
  " Block Method  
  CALL METHOD lo\_cust->block\_customer.



# Polymorphism

Polymorphism allows consumer to call several implementations with same name and interface parameters with same reference variable

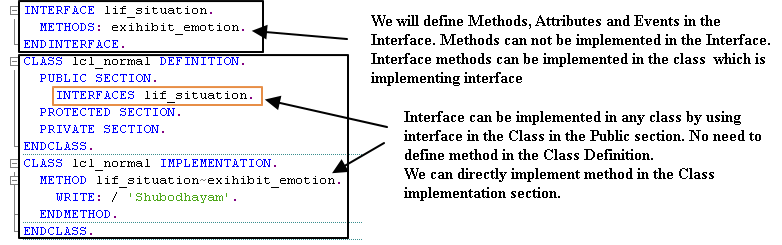


## Program

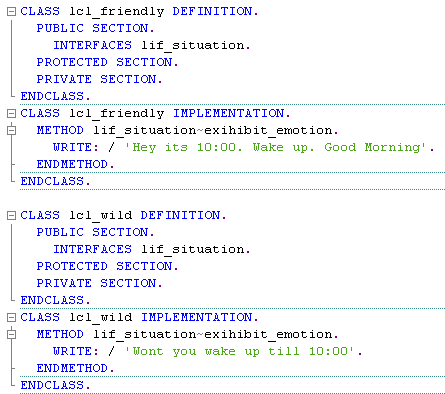
Interface Definition

Interface Implementation

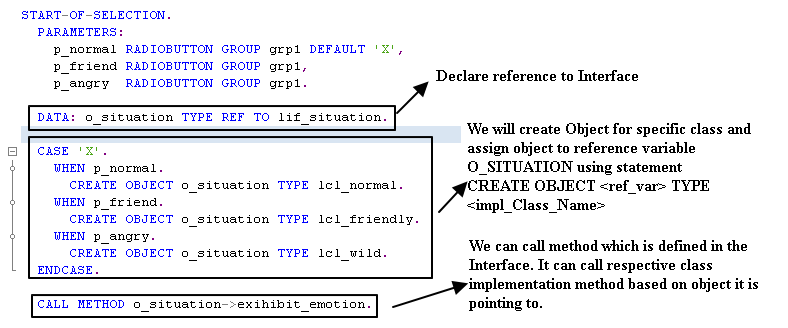
Using Interface in the Consumer section



Other Classes implemented the Interface



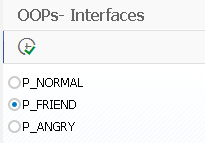
**Consuming the Classes**



**Complete Program**

\*&---------------------------------------------------------------------\*  
\*& Report ZTEST\_OOPS\_06  
\*&---------------------------------------------------------------------\*  
\*&  
\*&---------------------------------------------------------------------\*  
REPORT ztest\_oops\_06.  
  
INTERFACE lif\_situation.  
  METHODS: exihibit\_emotion.  
ENDINTERFACE.  
CLASS lcl\_normal DEFINITION.  
  PUBLIC SECTION.  
    INTERFACES lif\_situation.  
  PROTECTED SECTION.  
  PRIVATE SECTION.  
ENDCLASS.  
CLASS lcl\_normal IMPLEMENTATION.  
  METHOD lif\_situation~exihibit\_emotion.  
    WRITE: / 'Shubodhayam'.  
  ENDMETHOD.  
ENDCLASS.  
  
CLASS lcl\_friendly DEFINITION.  
  PUBLIC SECTION.  
    INTERFACES lif\_situation.  
  PROTECTED SECTION.  
  PRIVATE SECTION.  
ENDCLASS.  
CLASS lcl\_friendly IMPLEMENTATION.  
  METHOD lif\_situation~exihibit\_emotion.  
    WRITE: / 'Hey its 10:00. Wake up. Good Morning'.  
  ENDMETHOD.  
ENDCLASS.  
  
CLASS lcl\_wild DEFINITION.  
  PUBLIC SECTION.  
    INTERFACES lif\_situation.  
  PROTECTED SECTION.  
  PRIVATE SECTION.  
ENDCLASS.  
CLASS lcl\_wild IMPLEMENTATION.  
  METHOD lif\_situation~exihibit\_emotion.  
    WRITE: / 'Wont you wake up till 10:00'.  
  ENDMETHOD.  
ENDCLASS.  
  
START-OF-SELECTION.  
  PARAMETERS:  
    p\_normal RADIOBUTTON GROUP grp1 DEFAULT 'X',  
    p\_friend RADIOBUTTON GROUP grp1,  
    p\_angry  RADIOBUTTON GROUP grp1.  
  
  DATA: o\_situation TYPE REF TO lif\_situation.  
  CASE 'X'.  
    WHEN p\_normal.  
      CREATE OBJECT o\_situation TYPE lcl\_normal.  
    WHEN p\_friend.  
      CREATE OBJECT o\_situation TYPE lcl\_friendly.  
    WHEN p\_angry.  
      CREATE OBJECT o\_situation TYPE lcl\_wild.  
  ENDCASE.  
  
  CALL METHOD o\_situation->exihibit\_emotion.

Output:



Execute

