IBM Global Services

ABAP Objects

April,2004

Topics to cover

- ❑ Different approaches to Programming
- ❑ Class and objects
- ❑ Methods
- ❑ Constructor
- ❑ Inheritance
- ❑ Interface
- ❑ Friendship between Classes
- ❑ Events
- ❑ Class-Based Exceptions

Different approaches of Programming

- Unstructured Programming.
- Procedural Programming.
- Object Oriented Programming.

Unstructured Programming

```
report ysubdel.
DATA : sal type p decimals 2,
      itax type p decimals 2,
     net_sal type p decimals 2 .

 sal = 12000.
   IF sal lt 5000 .
    itax = 0.
   ELSE.
    itax = sal * '0.01'.
   ENDIF.
    net_sal = sal - itax.
   write:/5 sal , itax , net_sal.
sal = 3500.
 IF sal lt 5000 .
    itax = 0.
 ELSE.
    itax = sal * '0.01'.
 ENDIF.
    net_sal = sal - itax.
   write:/5 sal , itax , net_sal.
```

Characteristics

❖Consists of only one main program.

❖The program stands for a sequence of commands which modify data that is global throughout the whole program.

Disadvantages

❖Difficult to manage once the program becomes large.

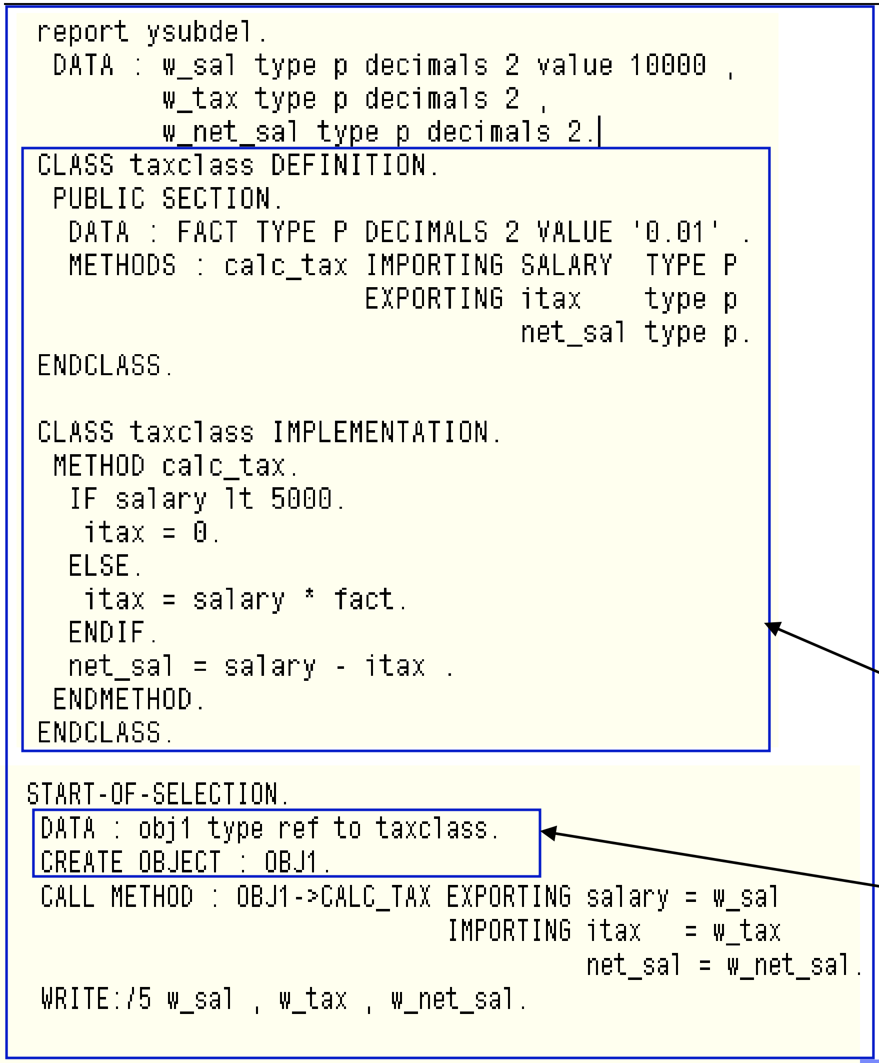
❖Same sequence of statements are repeated at multiple places, if they are needed at multiple locations.

Procedural Programming

```
report ysubdel.
DATA : sal  type p decimals 2 ,
       itax type p decimals 2 ,
      net_sal type p decimals 2.
  sal = 12000.
 PERFORM sub_calc_tax USING
      sal itax  net_sal.
  sal = 3500.
 PERFORM sub_calc_tax USING
    sal itax   net_sal.

FORM sub_calc_tax  USING
P_SAL P_ITAX P_NET_SAL.
IF p_sal lt 5000 .
 p_itax = 0.
ELSE.
 p_itax = sal * '0.01'.
ENDIF.
    p_net_sal = p_sal - p_itax.
ENDFORM.
```

- Programmer combines related sequences of statements into one single place, called procedure.
- A procedure call is used to invoke the procedure.
- After the sequence is processed, flow of control proceeds right after the position where the call was made.

Object Oriented Programming



```
report ysubdel.
 DATA : w_sal type p decimals 2 value 10000 ,
        w_tax type p decimals 2 ,
        w_net_sal type p decimals 2.
CLASS taxclass DEFINITION.
 PUBLIC SECTION.
  DATA : FACT TYPE P DECIMALS 2 VALUE '0.01' .
  METHODS : calc_tax IMPORTING SALARY  TYPE P
                     EXPORTING itax    type p
                               net_sal type p.
ENDCLASS.

CLASS taxclass IMPLEMENTATION.
 METHOD calc_tax.
  IF salary lt 5000.
   itax = 0.
  ELSE.
   itax = salary * fact.
  ENDIF.
  net_sal = salary - itax .
 ENDMETHOD.
ENDCLASS.

START-OF-SELECTION.
 DATA : obj1 type ref to taxclass.
 CREATE OBJECT : OBJ1.
 CALL METHOD : OBJ1->CALC_TAX EXPORTING salary = w_sal
                              IMPORTING itax    = w_tax
                                        net_sal = w_net_sal.
 WRITE:/5 w_sal , w_tax , w_net_sal.
```

❖ Classes and objects are used to model real world entity.

❖ Methods inside the classes perform the functions.

❖ Data used by the classes are protected between them.

Class defined and implemented

Object created from the class and used

Comparison between Procedural and Object Oriented Programming

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

Object Oriented Approach - key features

1. Better Programming Structure

2. Real world entity can be modeled very well

3.Stress on data security and access

4. Data encapsulation and abstraction

5. Reduction in code redundancy

Requirement for learning OOPS in ABAP

Adopt the universal approach for modern programming.

Know the latest ABAP technologies-BADI,workflow,XI, Custom controls etc

Use ABAP to integrate with Microsoft Technologies ,Java etc

Exploit class resources and functionalities provided by SAP.

Topics to cover

- ❑ Different approaches to Programming
- ❑ Class and objects
- ❑ Methods
- ❑ Constructor
- ❑ Inheritance
- ❑ Interface
- ❑ Friendship between Classes
- ❑ Events
- ❑ Class-Based Exceptions

Class and Object

Class is the template of an object. It is an abstract concept, which is realized by creating an object from it.

Still confused??? O.K ..see the example below

What is your idea about a Maruti-800 car? You will say that it is a four-wheel, five seater vehicle with a 800 cc engine.

The idea you have in your mind is the **class** for Maruti-800.Let us call it as class Maruti-800.

Now, you see a real Maruti-800 car in front of you. So, this is the **object** of class Maruti-800. In other words, the real car is realization/ instantiation of the class Maruti-800.

Attributes of Object Oriented Programming

- ❖Abstraction : Everything is visualized in terms of classes and objects.
- ❖Encapsulation : Necessary data are hidden from the users in class.
- ❖Inheritance : A class can be replicated into a subclass , which can be modified.
- ❖Polymorphism : Methods of same name behave differently in different classes.

Local and Global Classes

	Global Class	Local Class
Accessed by	Any program	Only the program where it is defined.
Stored in	In the Class Repository	Only in the program where it is defined.
Created by	Created using transaction SE24	Created using SE38
Namespace	Must begin with Y or Z	Can begin with any character

Defining Local Classes

```
REPORT  YSUBOOPS17  .
CLASS c1 DEFINITION.
PUBLIC SECTION.
 data : w_num type i value 5.
  methods : m1.
ENDCLASS.
CLASS c1 IMPLEMENTATION.
 METHOD M1.
  WRITE:/5 'I am M1 in C1'.
 ENDMETHOD.
ENDCLASS.
START-OF-SELECTION.
DATA : oref1 TYPE REF TO c1 .
CREATE OBJECT : oref1.
write:/5 oref1->w_num.
CALL METHOD : oref1->m1 .
```

❖Defined in the global area of a local program :-

CLASS <class name> DEFINITION.

…..

ENDCLASS.

❖All the attributes , methods, events and interfaces are declared here.

❖Cannot be declared inside a subroutine/function module.

❖Class definition cannot be nested.

Implementing Local Classes

```
REPORT  YSUBOOPS17   .
CLASS c1 DEFINITION.
PUBLIC SECTION.
 data : w_num type i value 5.
  methods : m1.
ENDCLASS.
CLASS c1 IMPLEMENTATION.
 METHOD M1.
  WRITE:/5 'I am M1 in C1'.
 ENDMETHOD.
ENDCLASS.
START-OF-SELECTION.
DATA : oref1 TYPE REF TO c1 .
CREATE OBJECT : oref1.
write:/5 oref1->w_num.
CALL METHOD : oref1->m1 .
```

➢Local class in a program is implemented as follows:-

```
     CLASS <class name> IMPLEMENTATION.
        …..
      ENDCLASS.
```

➢Methods used by the class are described here.

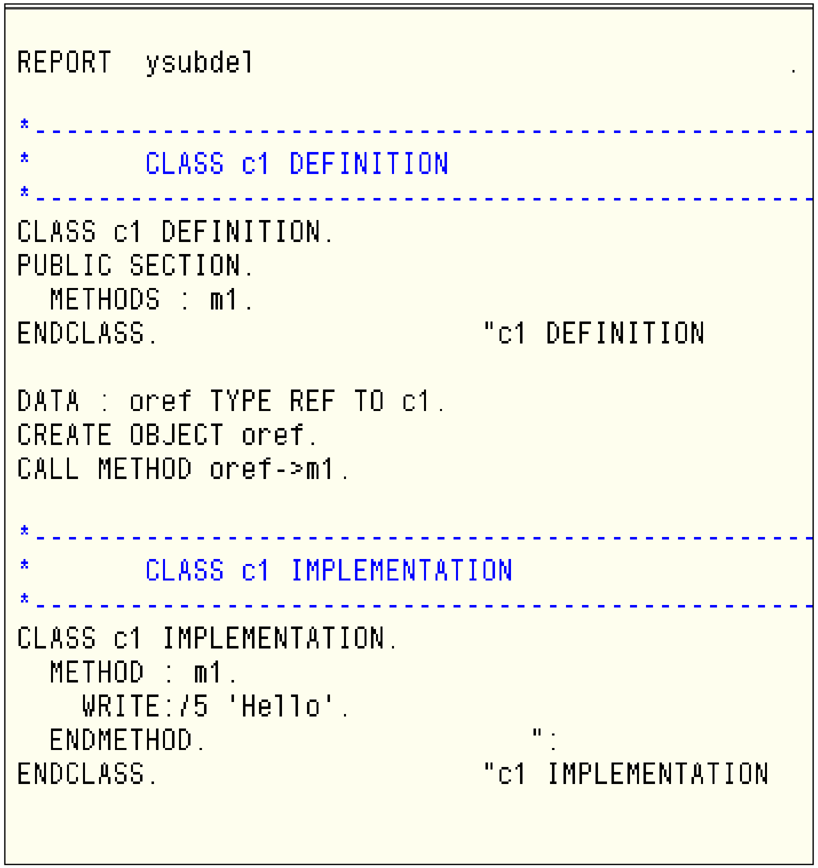
➢A class can be implemented

- ❖At the end of the program(like subroutines).
- ❖After the class definition.

If the latter is adopted, one must then assign subsequent non-declarative statements explicitly to a processing block, such as **START-OF-SELECTION**, so that they can be accessed.

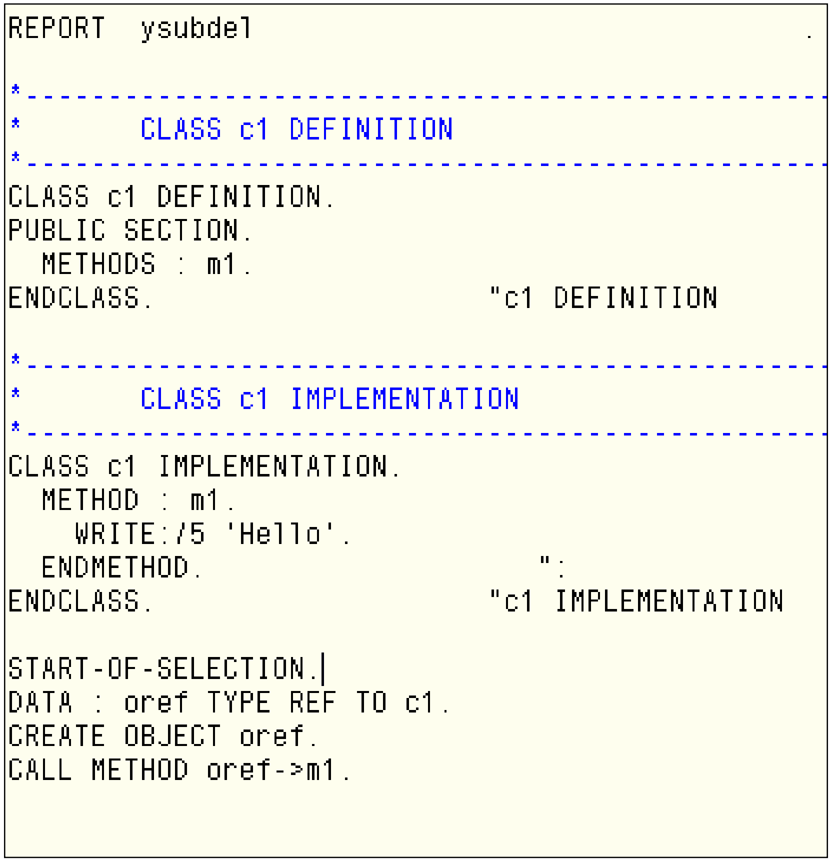
Different places of implementing class

Class implemented at the end of the program



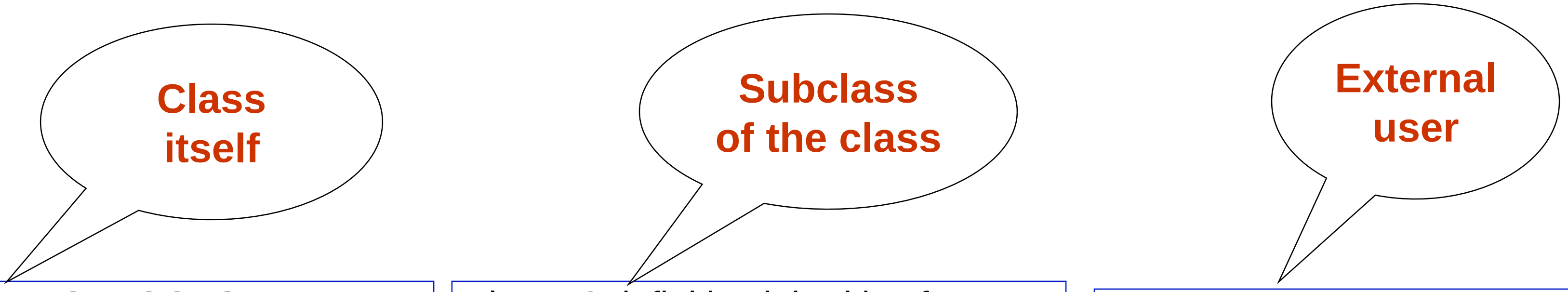
```
REPORT  ysubdel                                            .

*---------------------------------------------------------------
*       CLASS c1 DEFINITION
*---------------------------------------------------------------
CLASS c1 DEFINITION.
PUBLIC SECTION.
  METHODS : m1.
ENDCLASS.                    "c1 DEFINITION

DATA : oref TYPE REF TO c1.
CREATE OBJECT oref.
CALL METHOD oref->m1.

*---------------------------------------------------------------
*       CLASS c1 IMPLEMENTATION
*---------------------------------------------------------------
CLASS c1 IMPLEMENTATION.
  METHOD : m1.
    WRITE:/5 'Hello'.
  ENDMETHOD.                   ":
ENDCLASS.                    "c1 IMPLEMENTATION
```

Class implemented after Definition



```
REPORT  ysubdel                                            .

*---------------------------------------------------------------
*       CLASS c1 DEFINITION
*---------------------------------------------------------------
CLASS c1 DEFINITION.
PUBLIC SECTION.
  METHODS : m1.
ENDCLASS.                    "c1 DEFINITION

*---------------------------------------------------------------
*       CLASS c1 IMPLEMENTATION
*---------------------------------------------------------------
CLASS c1 IMPLEMENTATION.
  METHOD : m1.
    WRITE:/5 'Hello'.
  ENDMETHOD.                   ":
ENDCLASS.                    "c1 IMPLEMENTATION

START-OF-SELECTION.
DATA : oref TYPE REF TO c1.
CREATE OBJECT oref.
CALL METHOD oref->m1.
```

Who can use a class?



Class itself

```
REPORT  YSUBOOPS17  .
CLASS c1 DEFINITION.
 PUBLIC SECTION.
  data : w_num type i value 5.
  methods m1.
ENDCLASS.

CLASS c1 IMPLEMENTATION.
method m1.
 write:/5 ‘From class : ' , w_num.
endmethod.
ENDCLASS.
```

Subclass of the class

```
class c2 definition inheriting from
c1.
public section .
methods : m2.
endclass.

class c2 implementation.
method m2.
write:/5 ‘From subclass' , w_num .
endmethod.
endclass.
```

External user

```
START-OF-SELECTION.
DATA :
      oref1 TYPE REF TO c1 ,
      oref2 type ref to c2 .
CREATE OBJECT : oref1.
write:/5 ‘As an user ' ,
oref1->w_num.
Call method oref1->m1.
Call method oref2->m2.
```

Sections of a Class

❖There is no default visibility section in a class.

❖This sequence of visibility must be maintained in a class

```
CLASS C1 DEFINITION.
  PUBLIC SECTION.
    DATA:
    METHODS:
    EVENTS:
  PROTECTED SECTION.
    DATA:
    METHODS:
    EVENTS:
  PRIVATE SECTION.
    DATA:
    METHODS:
    EVENTS:
ENDCLASS.
```

A Class puts its components under three distinct sections:-

- Public Section:- Components placed here form the external interface of the class – they are visible to all users of the class as well as to methods within the class and to methods of subclasses*
- Protected Section:- Components placed in protected section are visible to the children of the class(subclass) as well as within the class
- Private Section:-Components placed here are accessible by the class itself.

A Silly Example

You are an employee (object of class:- employee). You have put your personal pens on the desk . These pens can be accesses by you(object of main class) , your son and daughters(sub-classes) as well as by any other people(users outside the class). So, your pens are in your **public** section.

You have a purse(attribute) in your pocket. Any other person (user outside the class) cannot use your purse(attribute) directly. But, your children(sub-classes of the class on which you are an object) can have full access to it.So, money in the purse is in your **protected** section.

You have a special skill which you have developed by putting constant effort for a long time. You can only access it. Neither your children(sub-classes of your class), nor any user have any access to it directly. So, you can consider that skill to be in your **private** section.

Components of a Class

A Class basically contains the following:-

❖**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.

❖**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

```
REPORT  YSUBOOPS17  .
CLASS c1 DEFINITION.
 PUBLIC SECTION.
  data : i_num type i value 5.
  class-data :
    s_num type i value 6 .
ENDCLASS.

CLASS c1 IMPLEMENTATION.
ENDCLASS.

START-OF-SELECTION.
DATA : oref1 TYPE REF TO c1 .
CREATE OBJECT : oref1.
write:/5 oref1->i_num.
write:/5 c1=>s_num .
write:/5 oref1->s_num.
```

- 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 ‘ **=>**’ .

Two Additions in Local Class Definition

Addition 1 : CLASS class DEFINITION DEFERRED.

Used to refer to a class at some point in a code and the class is not defined before the line.

```
CLASS C2 DEFINITION DEFERRED.
 CLASS C1 DEFINITION.
  PUBLIC SECTION.
    DATA O2 TYPE REF TO C2.
 ENDCLASS.
 CLASS C2 DEFINITION.
  public section.
    data : num type i value 5.
ENDCLASS.
```

```
start-of-selection.
 data : obj1 type ref to C1.
  CREATE OBJECT obj1.
  create object obj1->o2.
  write:/5 obj1->o2->num .
```

Two Additions in Local Class Definition

Addition 2 : CLASS class DEFINITION LOAD

The compiler normally loads the description of a **global** class from the **class library** the first time you use the class in your program . However, if the first access to a global class in a program is to its **static components** or in the definition of an **event handler method** , you must load it explicitly using the statement CLASS class DEFINITION LOAD. This variant has no corresponding ENDCLASS statement.

Creating an Object

```
report ysubdel.

CLASS c1 DEFINITION.
 PUBLIC SECTION.
  DATA : prog(4) type c value 'ABAP'.
ENDCLASS.

CLASS c1 IMPLEMENTATION.
ENDCLASS.

 START-OF-SELECTION.
  DATA : obj1 type ref to c1.
  CREATE OBJECT : OBJ1.        ◄──────────── Step 1
  write:/5 obj1->prog.   ◄─────────────────── Step 2
```

To create an object , the following steps need to be followed:-

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

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

Step 2 : Create an object from the reference variable:-

CREATE OBJECT <object name> .

CREATE PUBLIC|PROTECTED|PRIVATE ADDITIONS

Syntax : CLASS <classname> DEFINITION … [CREATE PUBLIC|PROTECTED|PRIVATE]

CREATE PUBLIC addition is implicit in every class definition if the other CREATE additions are not used. It defines the default state, that is, that every user can create instances of the class.

Addition of CREATE PROTECTED means the class can only be instantiated by itself or its subclass.

Addition of CREATE PRIVATE means the class can only instantiate itself or the friends of the class can instantiate it.

Attributes

Attributes are internal data fields within a class that can have any ABAP data type.

Instance Attributes

❖Exist **separately in each instance** (object) of the class .

❖Declared using the DATA statement.

❖One cannot use the TABLES or NODES statement in the declaration part of the class.

Static Attributes

❖**Available once for each class** .

❖Declared using the CLASS-DATA statement and are retained throughout the entire runtime. All the objects within a class can access its static attributes.

Attributes…continued

Types and Constants in Classes

Types

TYPES statement is used to define any number of your own ABAP data types within a class.

Constants

Constants are special static attributes, whose values are specified when they are declared and which cannot be changed later. The CONSTANTS statement is used to declare constants. Constants are not instance-specific - they are available once and once only for all the objects in the class. Constants declared inside a class can also be used outside the class, without even referring to it.

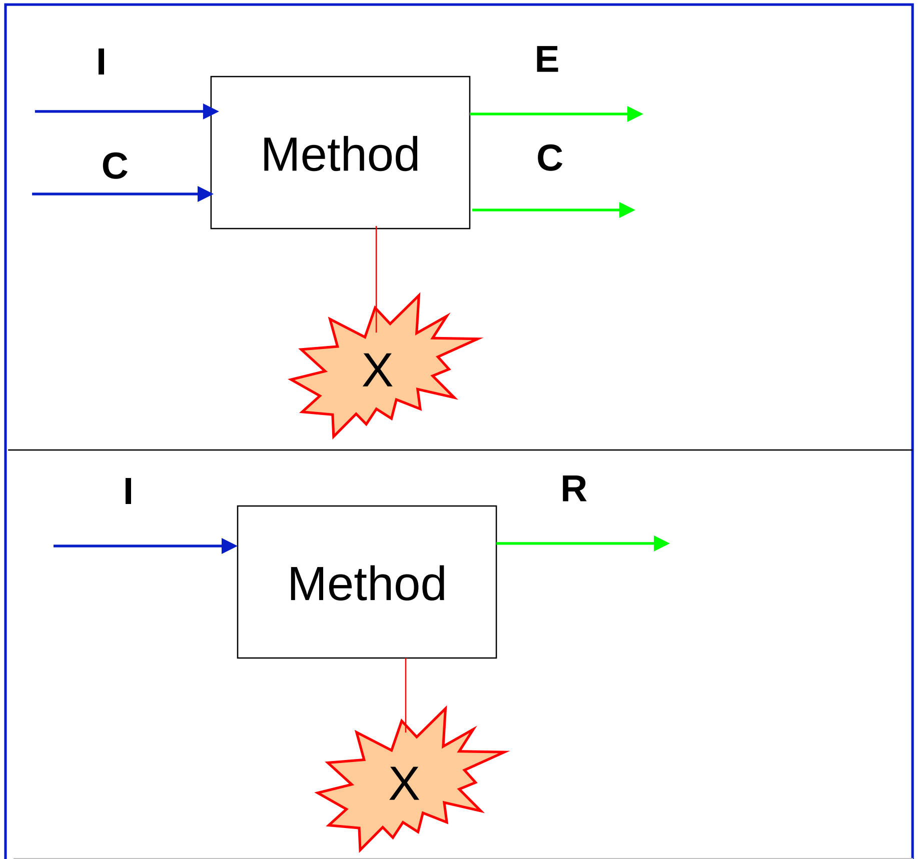
Read-only

Read-only data can be changed only by the methods of the class or its subclasses . But, the external users cannot change the data.

Topics to cover

- ❑ Different approaches to Programming
- ❑ Class and objects
- ❑ Methods
- ❑ Constructor
- ❑ Inheritance
- ❑ Interface
- ❑ Friendship between Classes
- ❑ Events
- ❑ Class-Based Exceptions

Methods



I – Import Parameters **E**-Export Parameters

C – Changing Parameters **R**- Returning Parameters

X - Exceptions

❖Methods are the internal procedures of a class .

❖Have a parameter interface, through which the system passes values to them when they are called, and through which they can return values to the caller.

❖Can access all the attributes of their class.

❖The private attributes of a class can only be changed using methods.

❖**One can draw the analogy between methods and function modules in ABAP**

Syntax for defining methods

- Declared in the definition part of a class or in an interface as follows:-

```
METHODS <meth> IMPORTING.. [VALUE(]<i_i>[)] TYPE type [OPTIONAL]..
               EXPORTING.. [VALUE(]<e_i>[)] TYPE type [OPTIONAL]..
               CHANGING.. [VALUE(]<c_i>[)] TYPE type [OPTIONAL]..
               RETURNING VALUE(<r>)
               EXCEPTIONS.. <e_i>..
```

- Default way of passing a parameter in a method is by reference.

- To pass a parameter by value, VALUE addition should be used.

- The return value (RETURNING parameter) must always be passed explicitly as a value. This is suitable for methods that return a single output value. If you use RETURNING, you cannot use EXPORTING or CHANGING parameters.

- An IMPORTING parameter transferred by reference, cannot be changed in the method.

- With function modules, one should type each parameter but are not forced to; with methods, you must type each parameter.

Typing Parameters

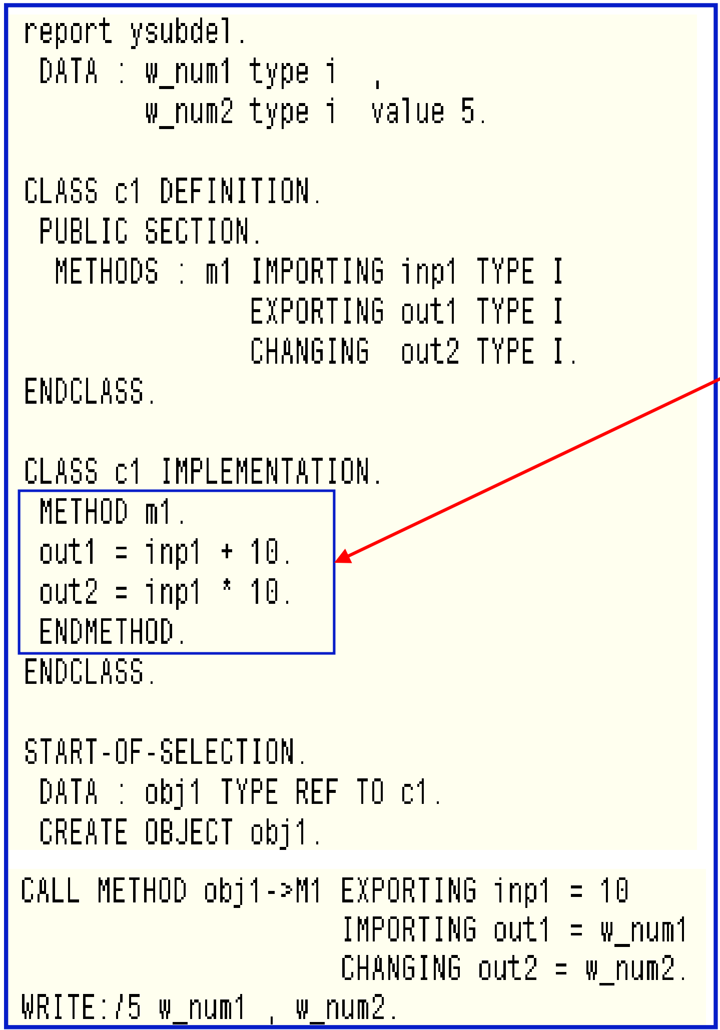
All parameters **must** be declared using the addition **TYPE** or the addition **LIKE**.

With **LIKE** you can **only** refer to data types of class attributes known at this point in **ABAP Objects** and not to data types from the ABAP Dictionary.

The following entries in the table shown below are allowed after **TYPE** as parameter types.

Syntax	Relevance
… TYPE t	stands for all data types from the ABAP Dictionary and all **ABAP** data types known at this point .
… TYPE ANY	stands explicitly for a parameter declaration without type reference. This means that the parameter inherits all technical attributes of the parameter that has been passed when the method is called.
… TYPE REF TO cif	stands for a class or interface reference.
… TYPE LINE OF itab	stands for the line type of an internal table **itab**.
TYPE [ANY|INDEX|STANDARD|SORTED|HASHED] TABLE	stands for the internal table type of the table type specified

Implementing Methods



```
report ysubdel.
 DATA : w_num1 type i  ,
        w_num2 type i  value 5.

CLASS c1 DEFINITION.
 PUBLIC SECTION.
  METHODS : m1 IMPORTING inp1 TYPE I
               EXPORTING out1 TYPE I
               CHANGING  out2 TYPE I.
ENDCLASS.

CLASS c1 IMPLEMENTATION.
 METHOD m1.
 out1 = inp1 + 10.
 out2 = inp1 * 10.
 ENDMETHOD.
ENDCLASS.

START-OF-SELECTION.
 DATA : obj1 TYPE REF TO c1.
 CREATE OBJECT obj1.

CALL METHOD obj1->M1 EXPORTING inp1 = 10
                     IMPORTING out1 = w_num1
                     CHANGING out2 = w_num2.
WRITE:/5 w_num1 , w_num2.
```

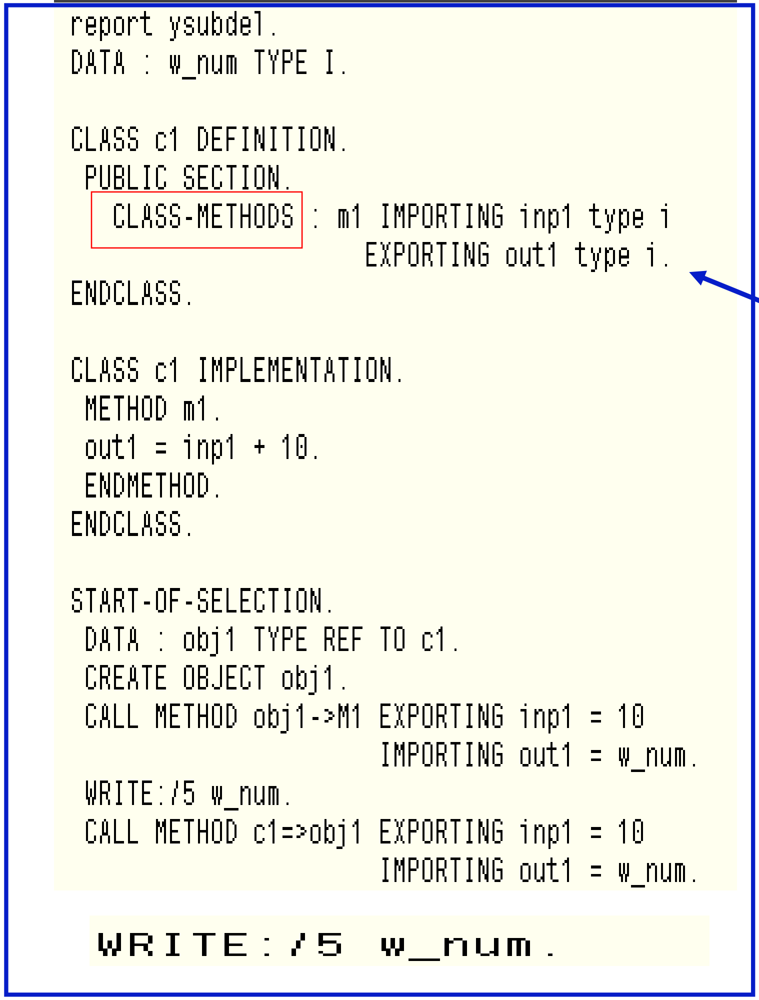
❑All methods declared in the definition part of a class should be implemented in the implementation section of the class within the following block:-

```
  METHOD <meth>.
   ...
 ENDMETHOD.
```

❑One should not specify any interface parameters at the time of implementation, since these are defined in the method declaration.

❑The interface parameters of a method behave like local variables within the method implementation. You can define additional local variables within a method using the DATA statement.

Static Method



```
report ysubdel.
DATA : w_num TYPE I.

CLASS c1 DEFINITION.
 PUBLIC SECTION.
   CLASS-METHODS : m1 IMPORTING inp1 type i
                      EXPORTING out1 type i.
ENDCLASS.

CLASS c1 IMPLEMENTATION.
 METHOD m1.
 out1 = inp1 + 10.
 ENDMETHOD.
ENDCLASS.

START-OF-SELECTION.
 DATA : obj1 TYPE REF TO c1.
 CREATE OBJECT obj1.
 CALL METHOD obj1->M1 EXPORTING inp1 = 10
                      IMPORTING out1 = w_num.
 WRITE:/5 w_num.
 CALL METHOD c1=>obj1 EXPORTING inp1 = 10
                      IMPORTING out1 = w_num.
```

```
WRITE:/5 w_num.
```

- Like any other class components , methods can be static or instance.
- To declare static methods, use the following statement:

  CLASS-METHODS <meth>...

- Static methods can have import and (export/ changing parameters) or returning parameters . It can also raise exceptions.
- Static methods can only work with the static attributes of your class. Instance methods can work with both static and instance attributes.
- One can call a static method using an object of the class or directly using the class by class component selector.

Calling Methods

```
REPORT  YSUBOOPS17  .
 DATA : w_num1 type i value 5,
       w_num3 type i value 7,
       w_num4 type i.

CLASS c1 DEFINITION.
 PUBLIC SECTION.
  METHODS : m1
             IMPORTING num1 TYPE i
             EXPORTING num4 TYPE i
             CHANGING  num3 type i.
ENDCLASS.
CLASS c1 IMPLEMENTATION.
method m1.
  num3 = num3 * 2.
  num4 = ( num1 + num3 ) ** 2.
endmethod.
ENDCLASS.
START-OF-SELECTION.
DATA : oref1 TYPE REF TO c1 .
CREATE OBJECT : oref1.
CALL METHOD oref1->m1 EXPORTING num1 = w_num1
                      IMPORTING num4 = w_num4
                      CHANGING  num3 = w_num3.
Write:/5 w_num1 , w_num3 , w_num4 .
```

CALL METHOD <meth>

EXPORTING... $<i_i> = .<f_i>...$
IMPORTING... $<e_i> = .<g_i>...$
CHANGING ... $<c_i> = .<f_i>...$
RECEIVING r = h
EXCEPTIONS... $<e_i> = rc_i...$

- All non-optional input parameters must be passed using the EXPORTING or CHANGING addition.
- One may import the output parameters into the program using the IMPORTING or RECEIVING addition.
- One may handle any exceptions triggered by the exceptions using the EXCEPTIONS addition.

Calling Methods…continued

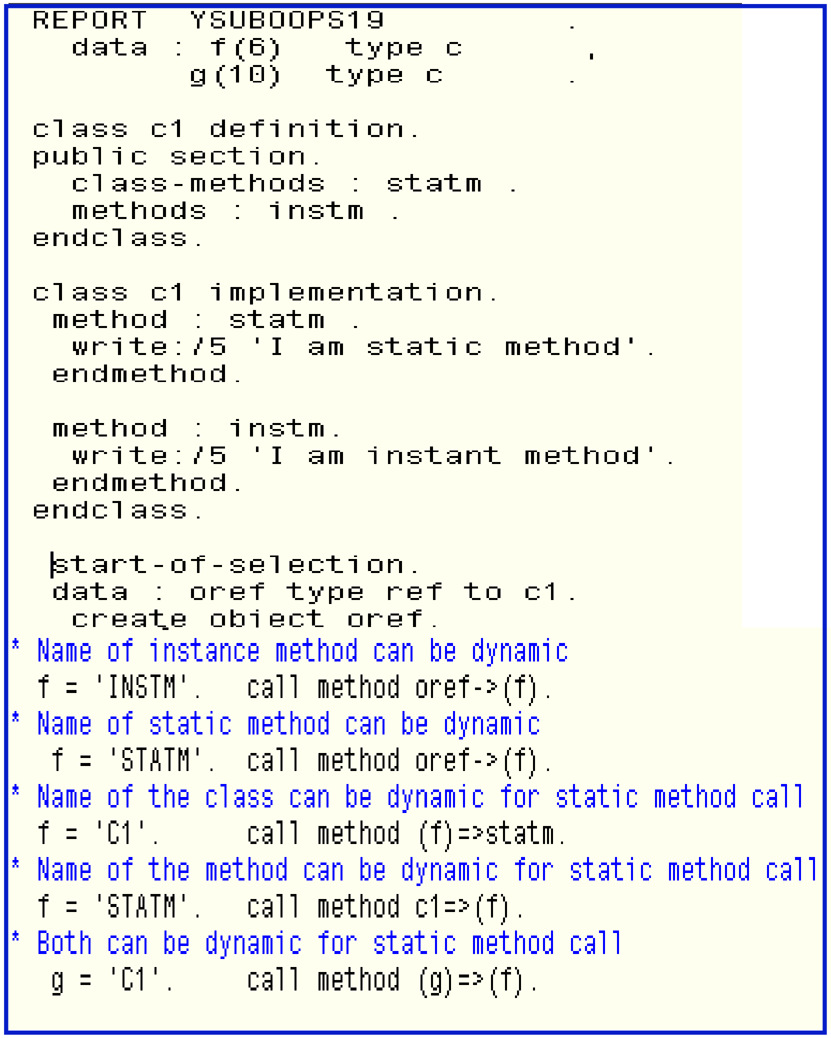
The way in which you address the method <method> depends on the method itself and from where you are calling it .

Syntax	Relevance
CALL METHOD <meth>...	Methods of the same class(in the implementation part) can be called directly using their name <meth>.
CALL METHOD <ref>-><meth>...	Visible instance methods can be called from outside the class using this syntax where <ref> is a reference variable whose value points to an instance of the class.
CALL METHOD <class>=><meth>...	. Visible static methods can be called from outside the class using this syntax where <class> is the name of the relevant class.
CALL METHOD <ref>-><meth>...	Visible instance methods can be called from outside the class using this syntax where <ref> is a reference variable whose value points to an instance of the class.
CALL METHOD <ref>-><meth>(f1 = a1 f2 = a2…….)	Used to call the method that consists only of IMPORTING parameters a1,a2 etc…

Calling Methods…continued

Syntax	Relevance
CALL METHOD <method>(f).	Used to call a method where :- ❖The interface of a method consists only of a single IMPORTING parameter ❖There is a single non-optional IMPORTING parameter and several other optional IMPORTING parameters. ❖All the IMPORTING parameters are optional , one being declared as PREFERRED PARAMETER.
f1=<ref>-><meth>(……..)	The method has one returning parameter whose values is transferred to the variable f1.
MOVE <ref>->meth(…) TO f1.	The method has one returning parameter whose values is transferred to the variable f1.

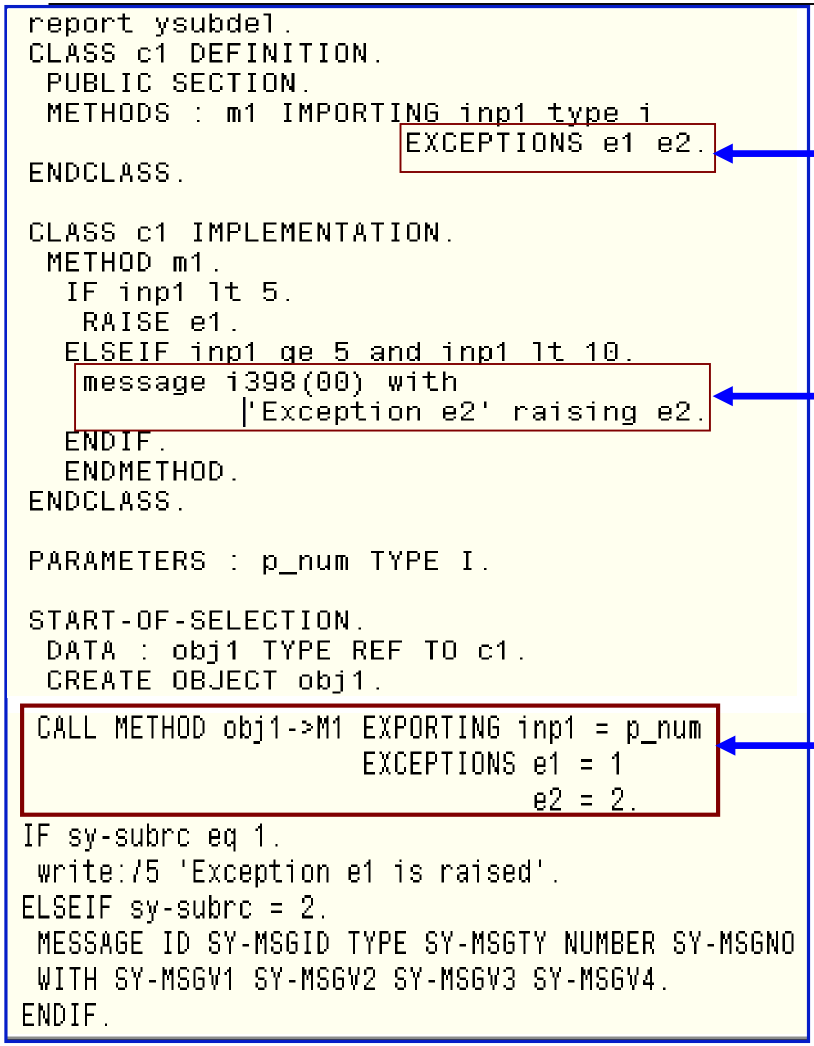
Dynamic Method Calls



```
REPORT  YSUBOOPS19          .
  data : f(6)   type c       ,
         g(10)  type c       .

class c1 definition.
public section.
  class-methods : statm .
  methods : instm .
endclass.

class c1 implementation.
 method : statm .
  write:/5 'I am static method'.
 endmethod.

 method : instm.
  write:/5 'I am instant method'.
 endmethod.
endclass.

 start-of-selection.
 data : oref type ref to c1.
  create object oref.
* Name of instance method can be dynamic
 f = 'INSTM'.   call method oref->(f).
* Name of static method can be dynamic
  f = 'STATM'.  call method oref->(f).
* Name of the class can be dynamic for static method call
 f = 'C1'.      call method (f)=>statm.
* Name of the method can be dynamic for static method call
 f = 'STATM'.   call method c1=>(f).
* Both can be dynamic for static method call
  g = 'C1'.     call method (g)=>(f).
```

- Instance, self-referenced, and static methods can all be called dynamically; the class name for static methods can also be determined dynamically:
  - oref->(method)
  - me->(method)
  - class=>(method)
  - (class)=>method
  - (class)=>(method)

Methods can raise exceptions



```
report ysubdel.
CLASS c1 DEFINITION.
 PUBLIC SECTION.
 METHODS : m1 IMPORTING inp1 type i
                    EXCEPTIONS e1 e2.
ENDCLASS.

CLASS c1 IMPLEMENTATION.
 METHOD m1.
  IF inp1 lt 5.
   RAISE e1.
  ELSEIF inp1 ge 5 and inp1 lt 10.
   message i398(00) with
            'Exception e2' raising e2.
  ENDIF.
  ENDMETHOD.
ENDCLASS.

PARAMETERS : p_num TYPE I.

START-OF-SELECTION.
 DATA : obj1 TYPE REF TO c1.
 CREATE OBJECT obj1.
 CALL METHOD obj1->M1 EXPORTING inp1 = p_num
                      EXCEPTIONS e1 = 1
                                 e2 = 2.
IF sy-subrc eq 1.
 write:/5 'Exception e1 is raised'.
ELSEIF sy-subrc = 2.
 MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
 WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ENDIF.
```

- Declare the name of the exceptions while defining the method in the class definition.
- Raise the exceptions within the method using any of the following syntaxes:-

  RAISE <exception name>

  MESSAGE …RAISING <exception>
- Handle the exceptions while calling method with different values of sy-subrc.

Parameter table

Instead of mentioning all the import,export, changing and receiving parameters for a dynamically calling method separately while calling it, one can use parameter table to specify information about the parameters and call the method.

The parameter table **itab** must be a hashed table of the table type ABAP_PARMBIND_TAB or of the line type ABAP_PARMBIND. These types are defined in the ABAP type group in the ABAP dictionary. The table has three columns:

❖**NAME** for the name of the formal parameter

❖**KIND** for the type of parameter passing

❖**VALUE** of the type **REF TO DATA** for the value of the actual parameter .

The type of parameter passing is defined for each formal parameter in the declaration of the called method. The content of the column KIND cannot therefore be initial. To check the type of parameter passing at runtime, one of the following constants from the global class CL_ABAP_OBJECTDESCR to the column KIND can be assigned:-

CL_ABAP_OBJECTDESCR=><xyz>

where <xyz> is :- EXPORTING/IMPORTING/CHANGING/RECEIVING for export,import, changing and receiving parameters respectively.

Parameter table…continued

If the specified and actual parameter types do not match each other, the system generates the exception CX_SY_DYN_CALL_ILLEGAL_TYPE, which can be handled.

For the value of the actual parameter, the reference VALUE of the table line must point to a data object that contains the required value. For this purpose, you can use the command **GET REFERENCEOF f INTO g**.

Exception table

Instead of dealing with each and every exception and assigning it to different sy-subrc values, one can use exception table to handle the exceptions when a method is called dynamically.

The exception table **itab** must be a hashed table of the table type **ABAP_EXCPBIND_TAB** or of the line type **ABAP_EXCPBIND**. These types are defined in the ABAP type group in the ABAP dictionary. The table has two columns:

- ❖**NAME** for the name of the exception
- ❖**VALUE** of type I for the **SY-SUBRC** value to be assigned

The column **NAME** is the unique table key. For each exception, exactly one line of the internal table can be filled. Here the numeric value is assigned to the component VALUE, which should be in **SY-SUBRC** after the exception has been triggered

Topics to cover

- ❑ Different approaches to Programming
- ❑ Class and objects
- ❑ Methods
- ❑ Constructor
- ❑ Inheritance
- ❑ Interface
- ❑ Friendship between Classes
- ❑ Events
- ❑ Class-Based Exceptions

Constructor

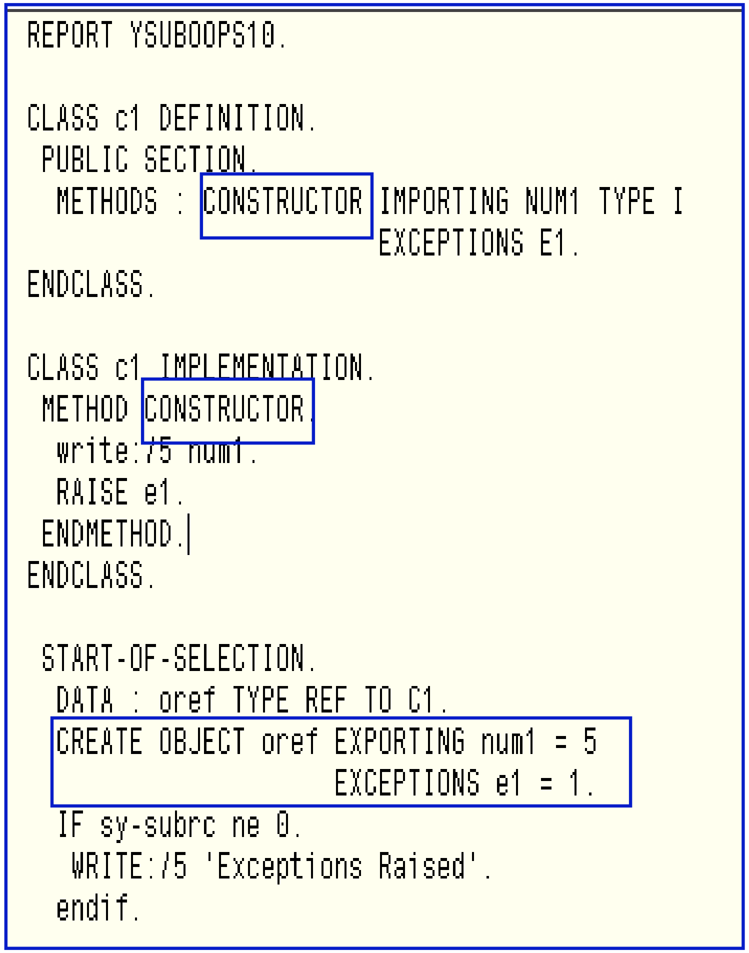
Constructors are special PUBLIC methods that are triggered when an object is instantiated from a class. They are necessary when you want to set the initial state of an object dynamically.

Like normal methods, there are two types of constructor - instance constructors and static constructors.

To use the instance constructor, the **CONSTRUCTOR** method must be declared in the public section of the class using the METHODS statement, and implemented in the implementation section.

Constructors must always be declared in the PUBLIC section of a class.

Instance Constructor



```
REPORT YSUBOOPS10.

CLASS c1 DEFINITION.
 PUBLIC SECTION.
  METHODS : CONSTRUCTOR IMPORTING NUM1 TYPE I
                        EXCEPTIONS E1.
ENDCLASS.

CLASS c1 IMPLEMENTATION.
 METHOD CONSTRUCTOR.
  write:/5 num1.
  RAISE e1.
 ENDMETHOD.
ENDCLASS.

 START-OF-SELECTION.
  DATA : oref TYPE REF TO C1.
  CREATE OBJECT oref EXPORTING num1 = 5
                     EXCEPTIONS e1 = 1.
  IF sy-subrc ne 0.
   WRITE:/5 'Exceptions Raised'.
  endif.
```

❖Executed once for each instance.

❖Called automatically, immediately after the CREATE OBJECT statement.

❖Can contain an interface with **IMPORTING** parameters and **EXCEPTIONS** , but **cannot have** any EXPORTING/CHANGING/RETURNING parameters .

❖The interfaces are defined using the same syntax as for normal methods in the METHODS statement. To transfer parameters and handle exceptions, use the **EXPORTING** and **EXCEPTIONS** additions to the **CREATE OBJECT** statement .

Static Constructor

```
REPORT  YSUBOOPS2.
CLASS c1 DEFINITION .
PUBLIC SECTION.
CLASS-DATA : NUM TYPE I VALUE 5.
CLASS-METHODS:CLASS_CONSTRUCTOR.
 ENDCLASS.

CLASS c1 IMPLEMENTATION.
METHOD CLASS_CONSTRUCTOR.
 WRITE:/5 'I am class
constructor'.
ENDMETHOD.
 ENDCLASS.

 START-OF-SELECTION.
   WRITE:/5 C1=>NUM.
```

❖Static methods, declared as **CLASS-METHODS :** **CLASS_CONSTRUCTOR** in the public section of the class definition and are also implemented in the implementation part.

❖**Has no interface parameters and cannot trigger exceptions**.

❖**Executed once in each program.** It is called automatically for the class **before it is accessed for the first time** - that is, before one of the following actions:

- **CREATE OBJECT obj** from the class.
- Call a static method : **[CALL METHOD] class=>meth**.
- Registering a static event handler method using **SET HANDLER class=>meth for obj**.
- Registering an event handler method for a static event of the class **class**.
- Addressing a static attribute with **class=>a**.

Static Constructor….continued

Exception:-

The static constructor is always called immediately before the action is executed, with one exception:

If the first access to the class is to address a static attribute, the static constructor is executed at the beginning of the processing block (dialog module, event block, procedure) in which the access occurs.

Caution:-

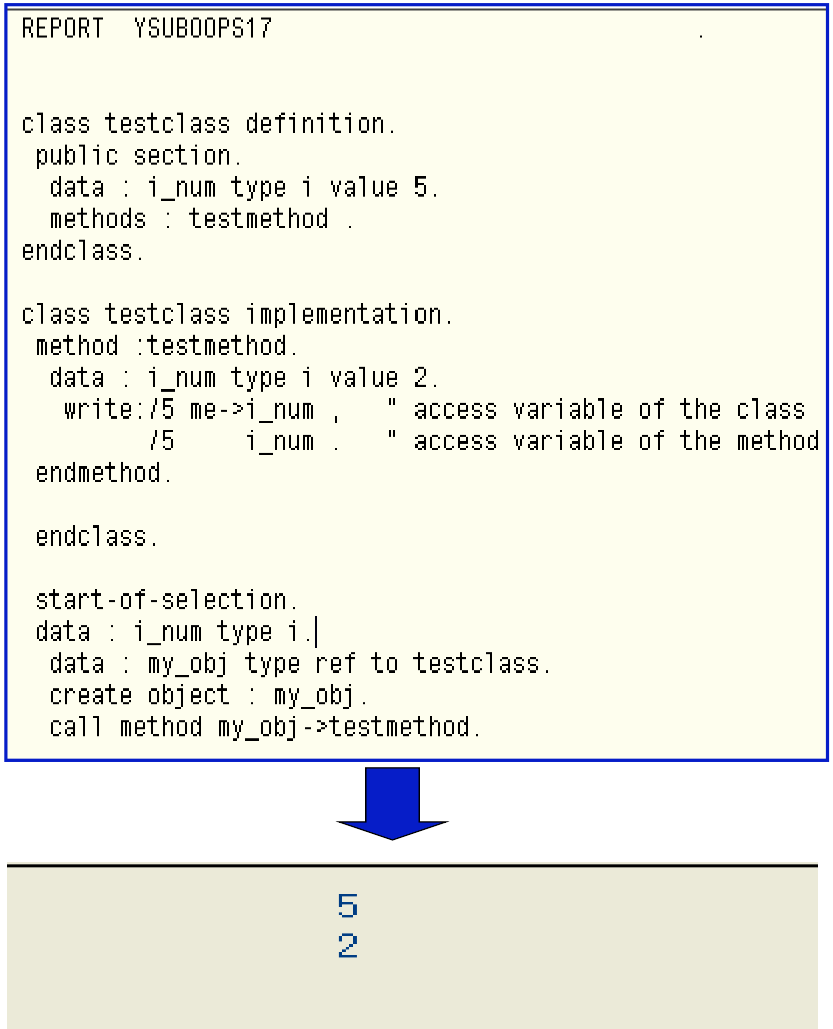
❖ The static constructor must not access its own class. Otherwise a runtime error will occur.

❖The point at which the static constructor is executed has not yet been finalized, and it may yet be changed. SAP only guarantees that it will be executed before the class is accessed for the first time. It is **recommended not to write programs that require the static constructor to be executed at the beginning of a processing block**.

Comparison between Different Methods

Type of Method	Triggering mechanism	Import parameter?	Export Parameter?	Changing Parameter?	Returning Parameter?	Can raise Exceptions?
Instance	Called by using object	Y	Y	Y	Y	Y
Static	Called by using object or referring to class	Y	Y	Y	Y	Y
Instance Constructor	Called automatically, immediately after the CREATE OBJECT statement.Called each time when an object is created.	Y	N	N	N	Y
Static Constructor	Executed once in each program. Called auto-matically for the class before it is accessed for the first time .	N	N	N	N	N

Self-Reference

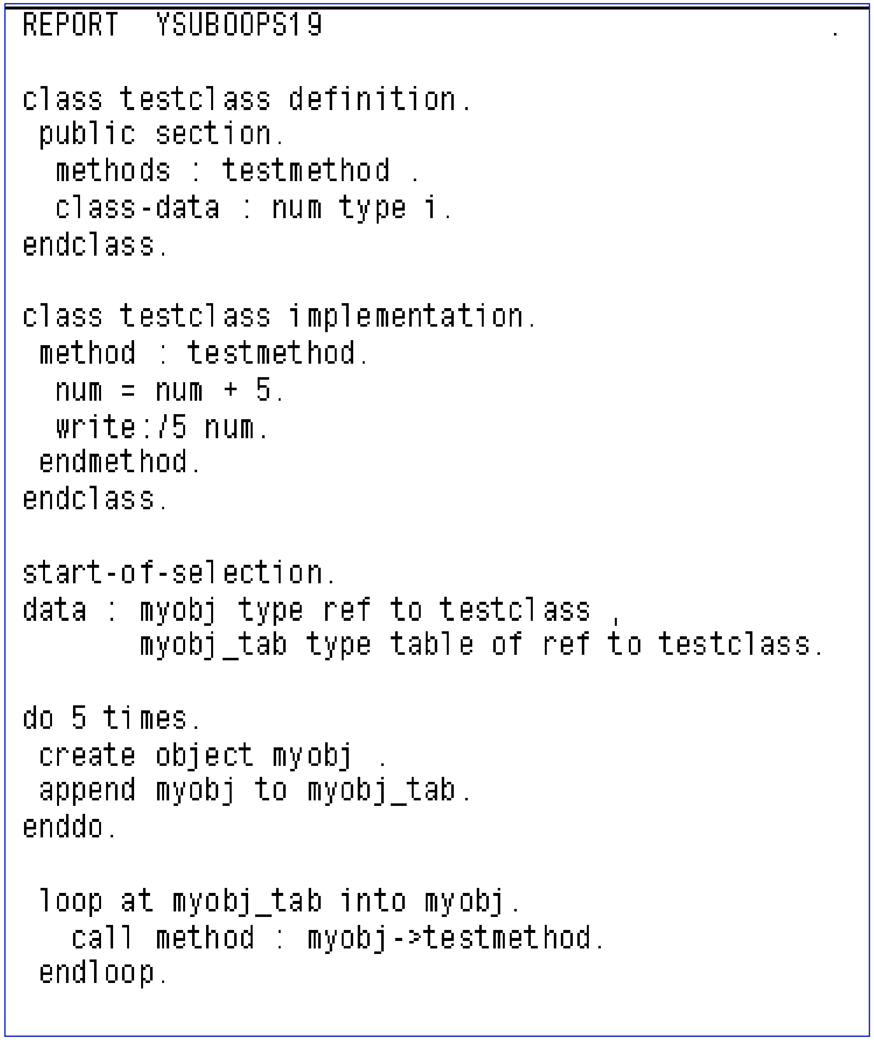


```
REPORT  YSUBOOPS17                                  .


class testclass definition.
 public section.
  data : i_num type i value 5.
  methods : testmethod .
endclass.

class testclass implementation.
 method :testmethod.
  data : i_num type i value 2.
   write:/5 me->i_num ,    " access variable of the class
         /5     i_num .    " access variable of the method
 endmethod.

 endclass.

 start-of-selection.
 data : i_num type i.
  data : my_obj type ref to testclass.
  create object : my_obj.
  call method my_obj->testmethod.
```

```
    5
    2
```

- Internally, each method has an implicit self-reference variable, the reserved word me
- A method can access the components of its class simply by their name; however,
  - It may use me simply for clarity
  - If a method declares a local variable with the same name as one of the class components, to avoid ambiguity it must use **me** to address the variable originally belonging to the class.
  - A method must use me to export a reference to itself (that is, its object)

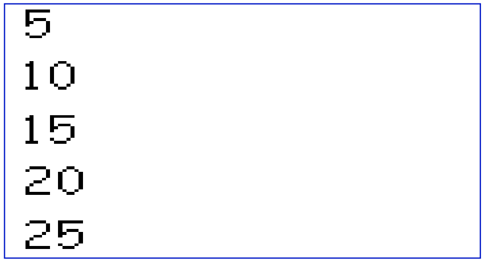
Pointer Tables



```
REPORT  YSUBOOPS19                                .

class testclass definition.
 public section.
  methods : testmethod .
  class-data : num type i.
endclass.

class testclass implementation.
 method : testmethod.
  num = num + 5.
  write:/5 num.
 endmethod.
endclass.

start-of-selection.
data : myobj type ref to testclass ,
       myobj_tab type table of ref to testclass.

do 5 times.
 create object myobj .
 append myobj to myobj_tab.
enddo.

 loop at myobj_tab into myobj.
   call method : myobj->testmethod.
 endloop.
```

- It is possible, in the same program, to instantiate many objects of the same class. However, in many situations internal tables are a more elegant way of creating a list of the same kind of data object
- Remember, reference variables are handled like any other data object with an elementary data type!

Output

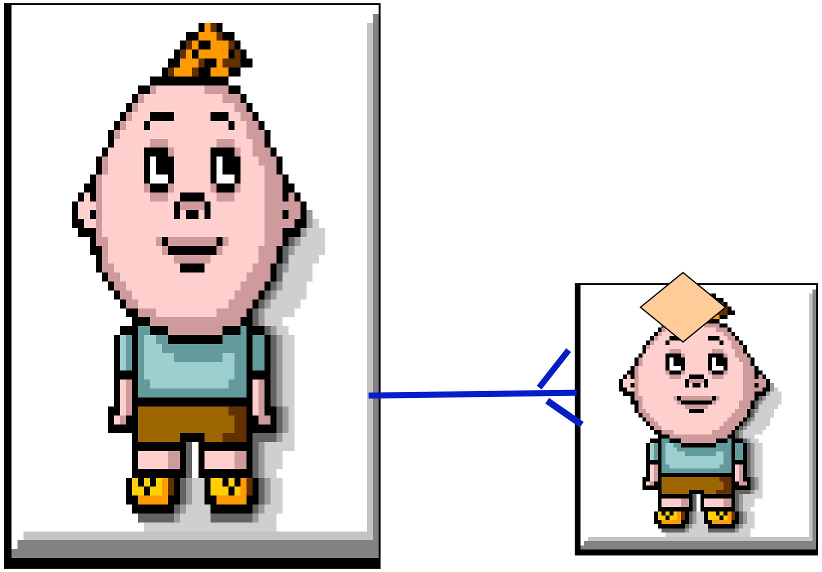


```
5
10
15
20
25
```

Topics to cover

- ❑ Different approaches to Programming
- ❑ Class and objects
- ❑ Methods
- ❑ Constructor
- ❑ Inheritance
- ❑ Interface
- ❑ Friendship between Classes
- ❑ Events
- ❑ Class-Based Exceptions

Inheritance



❖A single class can give birth to its subclasses by inheritance.

❖One parent class can have multiple subclasses, but one subclass can have only one parent class.Hence, **multiple inheritance is not allowed in ABAP**.

❖Subclasses have full access to all the public and protected components of parent class. They can modify to re-implement those components of their own.Also, subclasses can create new components in them.

❖Private components of parent classes are not known to subclasses. There can be components with same name in private section in both the parent and subclasses , each one working and being identified separately in their respective classes.

Visibility of Components

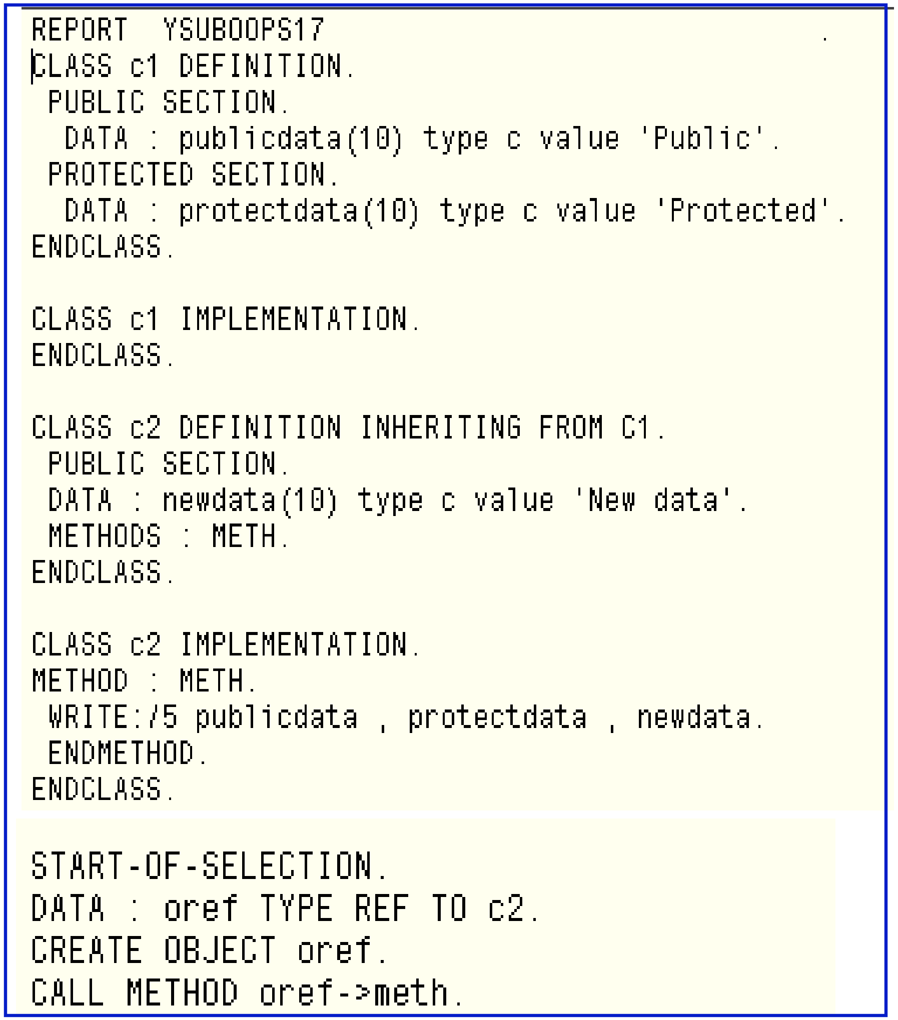
Each class component has a visibility. In ABAP Objects the whole class definition is separated into three visibility sections: PUBLIC, PROTECTED, and PRIVATE. One can never change component visibility via inheritance.

PUBLIC: This section is made up of the subclass' own public components together with all public components of all superclasses. There is unrestricted access to the public section of the subclass.

PROTECTED: This section is made up of the subclass' own protected components together with all protected components of all superclasses .

PRIVATE: This section is made up of the subclass' own private components accessible only in that subclass. Each subclass works with its own private components and can't even use the private components of its superclasses. But as long as a method inherited from a superclass is not redefined, it still uses the private attributes of the superclass, not those of the subclass, even if the subclass has private attributes of the same name.

Creating Subclass



```
REPORT  YSUBOOPS17                                    .
CLASS c1 DEFINITION.
 PUBLIC SECTION.
  DATA : publicdata(10) type c value 'Public'.
 PROTECTED SECTION.
  DATA : protectdata(10) type c value 'Protected'.
ENDCLASS.

CLASS c1 IMPLEMENTATION.
ENDCLASS.

CLASS c2 DEFINITION INHERITING FROM C1.
 PUBLIC SECTION.
 DATA : newdata(10) type c value 'New data'.
 METHODS : METH.
ENDCLASS.

CLASS c2 IMPLEMENTATION.
METHOD : METH.
 WRITE:/5 publicdata , protectdata , newdata.
 ENDMETHOD.
ENDCLASS.

START-OF-SELECTION.
DATA : oref TYPE REF TO c2.
CREATE OBJECT oref.
CALL METHOD oref->meth.
```

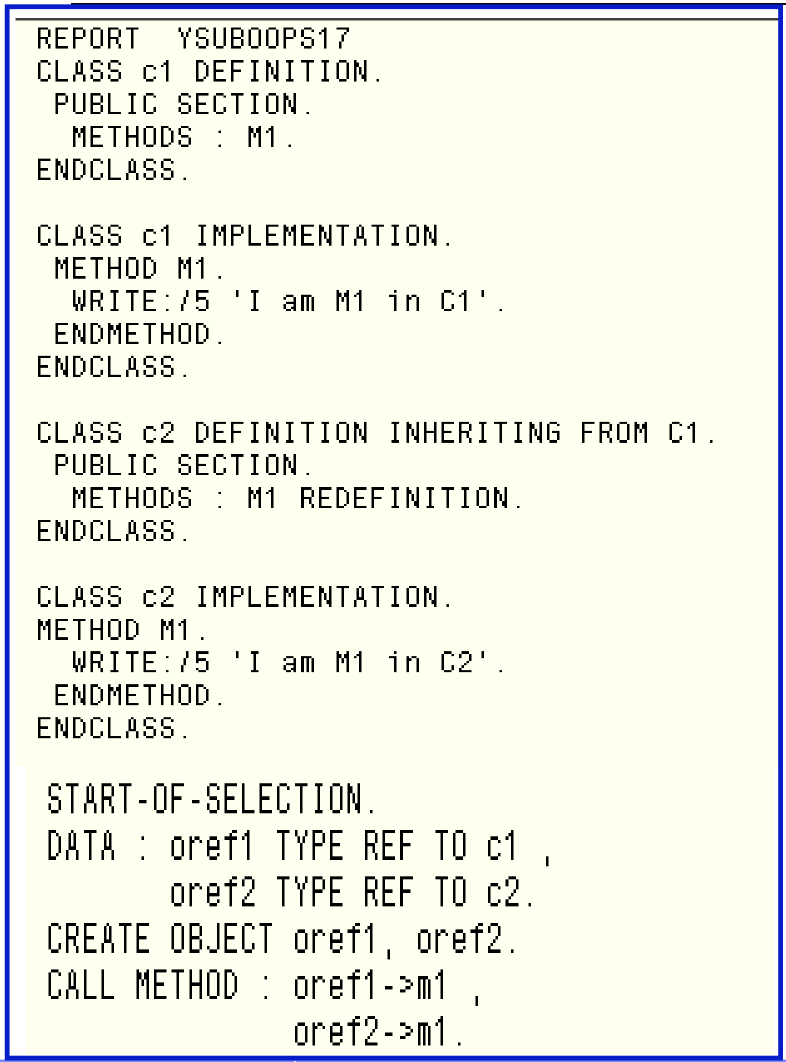
❖Subclasses can be created from its superclass using the syntax:-

CLASS <subclass> DEFINITION INHERITING FROM <superclass>.

❖Subclass inherits all the public and protected components of the superclass.

❖Superclass should not be declared as a FINAL class.

Modifying methods in subclass



```
REPORT  YSUBOOPS17
CLASS c1 DEFINITION.
 PUBLIC SECTION.
  METHODS : M1.
ENDCLASS.

CLASS c1 IMPLEMENTATION.
 METHOD M1.
  WRITE:/5 'I am M1 in C1'.
 ENDMETHOD.
ENDCLASS.

CLASS c2 DEFINITION INHERITING FROM C1.
 PUBLIC SECTION.
  METHODS : M1 REDEFINITION.
ENDCLASS.

CLASS c2 IMPLEMENTATION.
METHOD M1.
  WRITE:/5 'I am M1 in C2'.
 ENDMETHOD.
ENDCLASS.

START-OF-SELECTION.
DATA : oref1 TYPE REF TO c1 ,
       oref2 TYPE REF TO c2.
CREATE OBJECT oref1, oref2.
CALL METHOD : oref1->m1 ,
              oref2->m1.
```

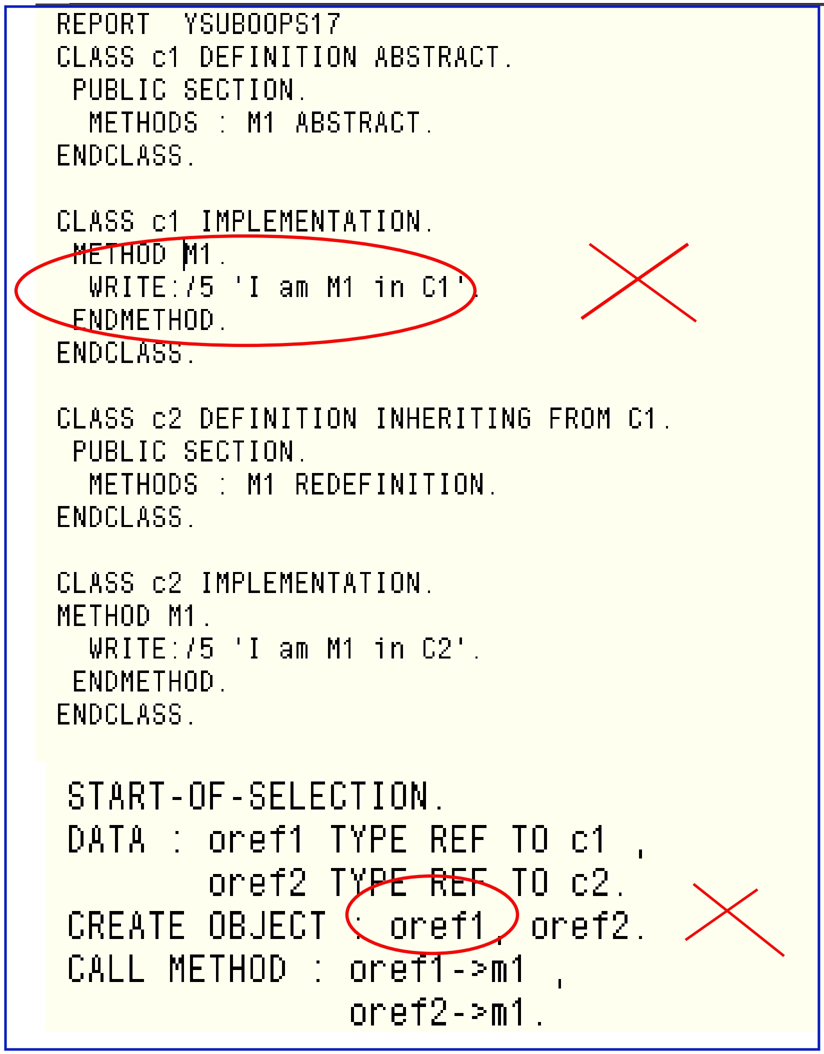
❖To redefine a public/protected method of a superclass in one of its subclasses, use the syntax in the subclass definition:-

METHOD <method name> REDEFINITION

❖The **interface and visibility of a method cannot be changed while redefining it.**

❖The method declaration and implementation in the superclass is not affected when you redefine the method in a subclass.

Abstract Methods and Classes



```
REPORT  YSUBOOPS17.
CLASS c1 DEFINITION ABSTRACT.
 PUBLIC SECTION.
  METHODS : M1 ABSTRACT.
ENDCLASS.

CLASS c1 IMPLEMENTATION.
 METHOD M1.
  WRITE:/5 'I am M1 in C1'.
 ENDMETHOD.
ENDCLASS.

CLASS c2 DEFINITION INHERITING FROM C1.
 PUBLIC SECTION.
  METHODS : M1 REDEFINITION.
ENDCLASS.

CLASS c2 IMPLEMENTATION.
METHOD M1.
  WRITE:/5 'I am M1 in C2'.
 ENDMETHOD.
ENDCLASS.

START-OF-SELECTION.
DATA : oref1 TYPE REF TO c1 ,
       oref2 TYPE REF TO c2.
CREATE OBJECT : oref1, oref2.
CALL METHOD : oref1->m1 ,
              oref2->m1.
```

One cannot create an object from an abstract class. Only subclasses can be derived from them.

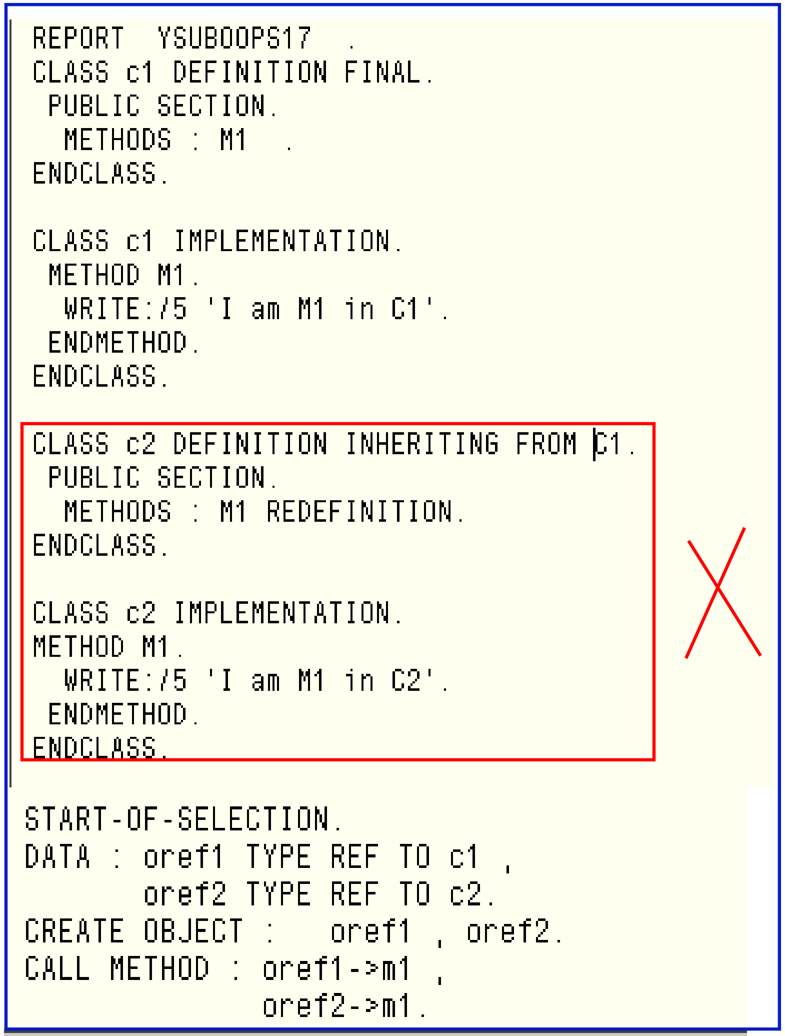
CLASS <classname> DEFINITION ABSTRACT.

Abstract methods cannot be implemented in the same class. Only the subclasses of that class can implement it.

METHODS <method_name> ….ABSTRACT

Any class containing an abstract method has to be an abstract class. All subsequent subclasses that do not implement the method must also be abstract. To implement an abstract method in a subclass, one need to redefine this subclass using the **REDEFINITION** addition.

Final Methods and Classes



```
REPORT  YSUBOOPS17  .
CLASS c1 DEFINITION FINAL.
 PUBLIC SECTION.
  METHODS : M1   .
ENDCLASS.

CLASS c1 IMPLEMENTATION.
 METHOD M1.
  WRITE:/5 'I am M1 in C1'.
 ENDMETHOD.
ENDCLASS.

CLASS c2 DEFINITION INHERITING FROM C1.
 PUBLIC SECTION.
  METHODS : M1 REDEFINITION.
ENDCLASS.

CLASS c2 IMPLEMENTATION.
METHOD M1.
  WRITE:/5 'I am M1 in C2'.
 ENDMETHOD.
ENDCLASS.

START-OF-SELECTION.
DATA : oref1 TYPE REF TO c1 ,
       oref2 TYPE REF TO c2.
CREATE OBJECT :   oref1 , oref2.
CALL METHOD : oref1->m1 ,
              oref2->m1.
```

Final classes cannot have subclasses. Only the class can be instantiated.

CLASS <classname> DEFINITION FINAL.

A final method cannot be redefined in subclasses

METHODS <method_name> ….FINAL

Inheritance and Static Attributes

```
CLASS c1 DEFINITION.
 PUBLIC SECTION .
  CLASS-DATA : num TYPE I VALUE 5 .
ENDCLASS.

 CLASS c1 IMPLEMENTATION.
 ENDCLASS.

CLASS c2 DEFINITION INHERITING FROM c1.
ENDCLASS.

CLASS c2 IMPLEMENTATION.
ENDCLASS.

 START-OF-SELECTION.
   c2=>num = 7.
  write:/5 c1=>num .
```

- Static attributes only exist once in each inheritance tree. One can change them from outside the class using the class component selector with any class name, or within any class in which they are shared.
- They are visible in all classes in the inheritance tree.

Output : 7

Inheritance and Instance Constructors

Superclasses and/or subclasses can have explicit constructors of their own. Constructor of a subclass sometimes have to call the constructor of the superclass using : **CALL METHOD : SUPER->CONSTRUCTOR** depending on the following:-

Case	Description	Necessity of calling constructor of superclass by subclass
1	None of the superclass and subclass have explicit constructor.	Not required
2	Superclass have explicit constructor, but subclass does not have any explicit constructor.	Not required
3	Superclass does not have an explicit constructor, but subclass have one.	Required
4	Both the superclass and subclass have explicit constructor	Required

Inheritance and Static Constructors

Every class has a static constructor called **CLASS_CONSTRUCTOR**.

The first when a subclass in a program is accessed, its static constructor is executed. But, before it can be executed, the static constructors of all of its superclasses must already have been executed. A static constructor may only be called once per program. Therefore, when one first address a subclass, the system looks for the next-highest superclass whose static constructor has not yet been executed. It executes the static constructor of that class, followed by those of all classes between that class and the subclass that is addressed.

Inheritance and Instance Creation

For each **individual class**, the **CREATE PUBLIC|PROTECTED|PRIVATE** additions to the CLASS statement control who can create an instance of the class or, in other words, can call its instance constructor.

There are several cases which needs to be analyzed to understand:-

Superclass with no addition or CREATE PUBLIC

Subclasses can have every **CREATE** addition. Without addition they inherit the attribute **CREATE PUBLIC**. The superclass instance constructor is visible to everyone. The subclass controls the visibility of its own instance constructor, independently of the superclass.

Inheritance and Instance Creation

1. Superclass with addition CREATE PROTECTED

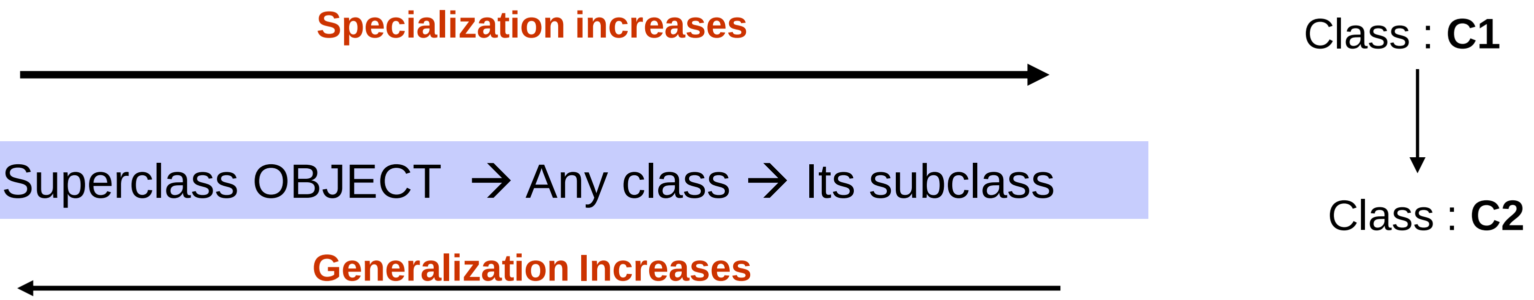
Subclasses can have every CREATE addition. Without addition they inherit the attribute CREATE PROTECTED. The superclass allows its subclasses unlimited instantiation and therefore also the publishing of its protected instance constructor.

1. Superclass with addition CREATE PRIVATE

If the subclass is **not a friend** of super class , the subclass cannot be instantiated. None of the **CREATE** additions is permitted with the subclass, because this would always lead to the unauthorized publishing of the superclass constructor.

If the subclass **is a friend** of super class, all **CREATE** additions are permitted with the subclass. The default one is CREATE PRIVATE. As a friend, the subclass can publish the superclass' private constructor in any form.

Static Type and Dynamic Type of a Variable



Specialization increases

Superclass OBJECT → Any class → Its subclass

Generalization Increases

Class : **C1**

Class : **C2**

The **static** type is the type (class or interface) with which the reference variable has been typed.

DATA : oref type ref to c1. “Static type of oref is pointing to class c1

The **dynamic** type is the class of the object to which the reference in a reference variable points .

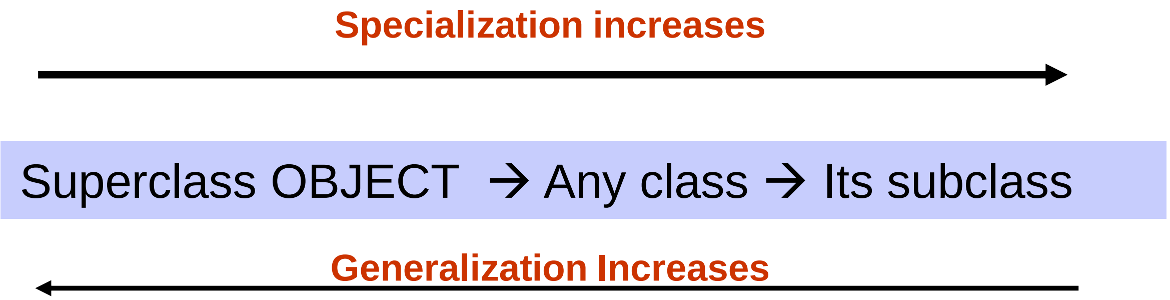
CREATE OBJECT OREF . “Dynamic type also points to c1

CREATE OBJECT OREF TYPE C2. “Dynamic type points to subclass c2 of C1

If the static and dynamic types of a reference variable are different, the **static type should always be more general than the dynamic type**.

.

Assignment between Reference Variables



Specialization increases

Superclass OBJECT → Any class → Its subclass

Generalization Increases

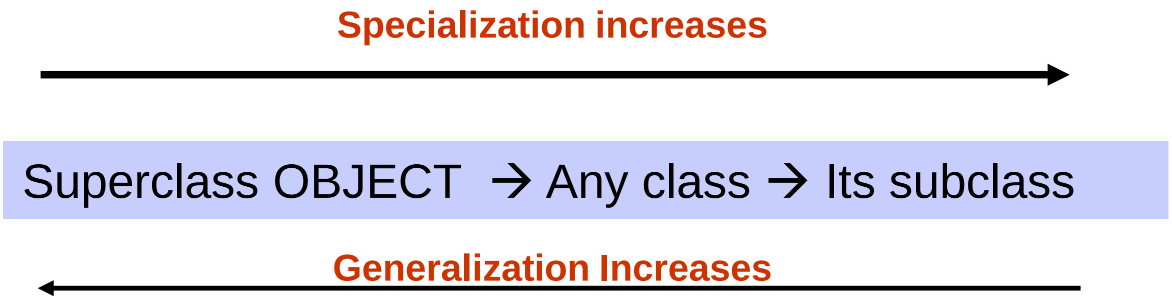
If you make assignments between reference variables, **the static type of the target variable must always be same or more generic than the dynamic type of the source variable.**

For example, a super class is always more generic than its subclass.

Syntax check for the program ensures that static type of target variable is more general than static type of source variable.

Narrowing Cast

The static type of the target variable must always be more generic than the static type of the source variable.



Specialization increases

Superclass OBJECT → Any class → Its subclass

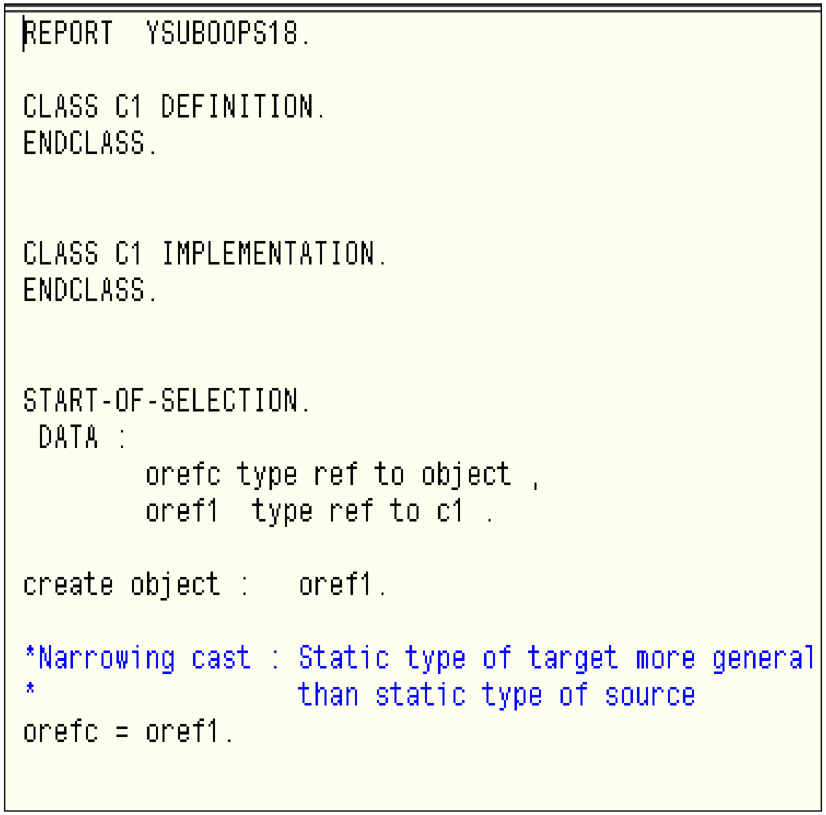
Generalization Increases

“Narrowing” cast means that the assignment changes from a more specialized view (with visibility to more components) to a more generalized view (with visibility to fewer components).

“**Narrowing cast**” is also referred to as “up cast” . “Up cast” means that the static type of the target variable can only change to higher nodes from the static type of the source variable in the inheritance tree, but not vice versa.

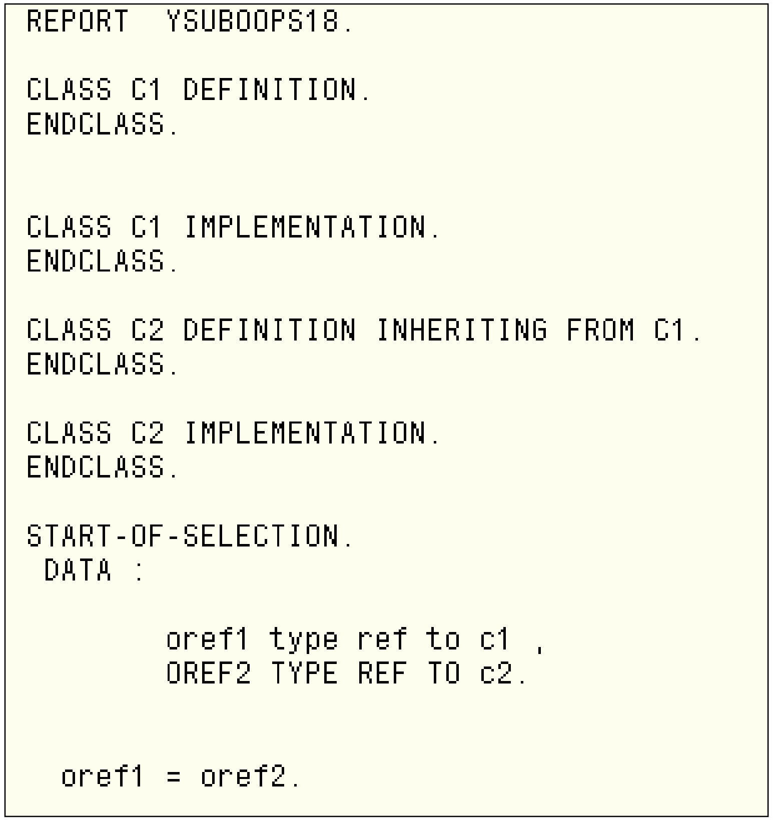
Example of Narrowing Cast

Reference variable of a class assigned to reference variable of class : object



```
REPORT  YSUBOOPS18.

CLASS C1 DEFINITION.
ENDCLASS.


CLASS C1 IMPLEMENTATION.
ENDCLASS.


START-OF-SELECTION.
 DATA :
        orefc type ref to object ,
        oref1  type ref to c1 .

create object :   oref1.

*Narrowing cast : Static type of target more general
*                 than static type of source
orefc = oref1.
```

Reference variable of a subclass assigned to reference variable of its superclass.



```
REPORT  YSUBOOPS18.

CLASS C1 DEFINITION.
ENDCLASS.


CLASS C1 IMPLEMENTATION.
ENDCLASS.

CLASS C2 DEFINITION INHERITING FROM C1.
ENDCLASS.

CLASS C2 IMPLEMENTATION.
ENDCLASS.

START-OF-SELECTION.
 DATA :

        oref1 type ref to c1 ,
        OREF2 TYPE REF TO c2.


  oref1 = oref2.
```

Widening Cast

```
DATA:   o_ref1 TYPE REF TO object,
        o_ref2 TYPE REF TO class.
...
o_ref2 ?= o_ref1.
...
CATH SYSTEM-EXCEPTIONS
move_cast_error = 4.
   o_ref2 ?= o_ref1.
ENDCATCH.
```

- In some cases, you may wish to make an assignment in which the static type of the target variable is less general than the static type of the source variable. This is known as a **widening cast**.
- The result of the assignment must still adhere to the rule that **the static type of the target variable must be the same or more general than the dynamic type of the source variable**.
- However, this can only be checked during runtime. To avoid the check on static type, use the special **casting operator** “?=“ and catch the potential runtime error **move_cast_error**

Polymorphism via Inheritance

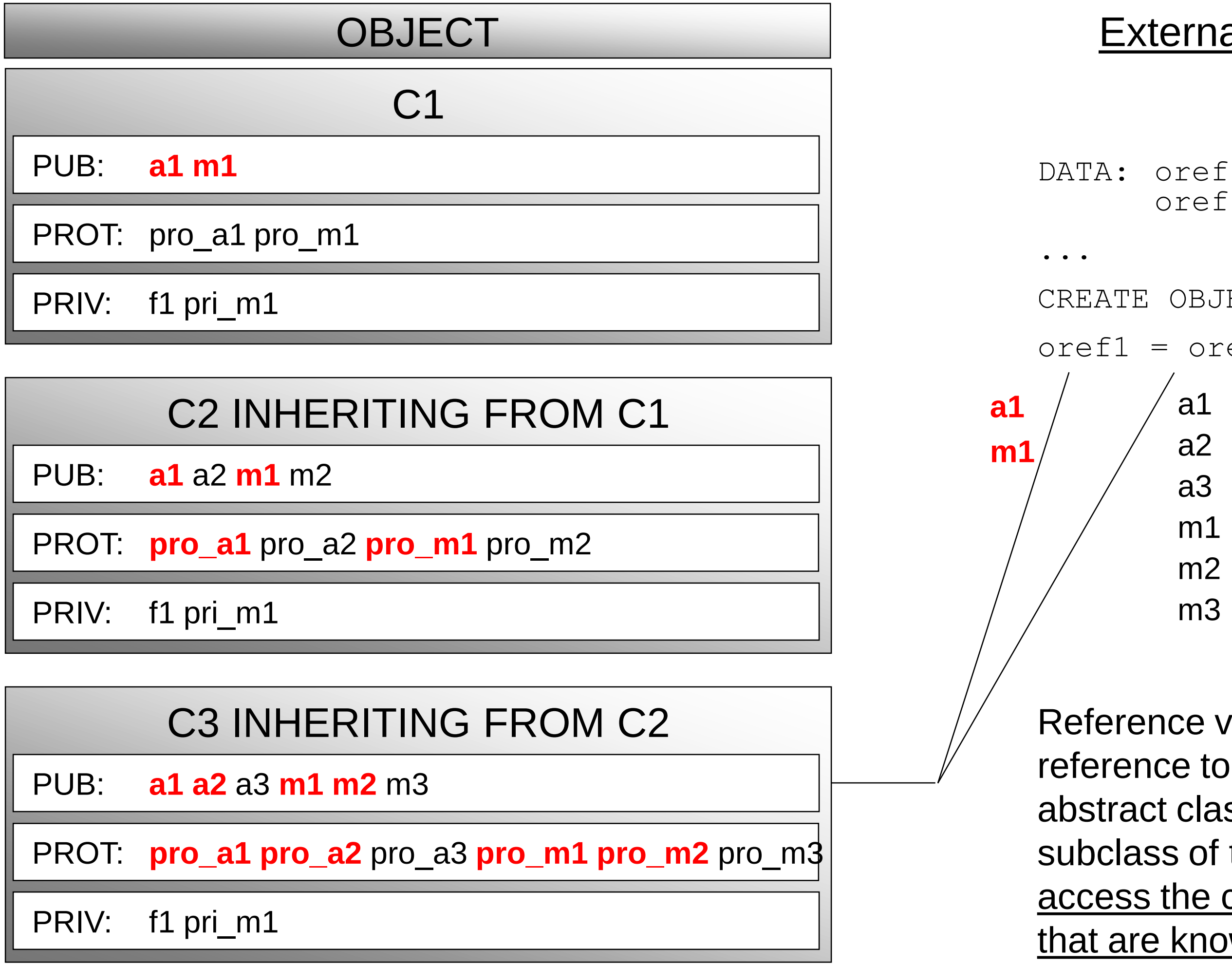
```
class c1 definition.
 . . . . . . .
endclass.

class c1 implementation.
. . . . . .
endclass.

class c2 definition inheriting from c1.
 . . . . . .
endclass.
class c2 implementation.
 . . . . . . .
endclass.
 start-of-selection.
  data : oref1 type ref to c1,
         oref11 type ref to c1,
         oref2 type ref to c2.
 create object oref1 type c2 .
 create object oref2.
 oref11 = oref2.
  write:/5 oref1->num ,
           oref11->num .
```

❖With inheritance, a reference variable defined with respect to a class may not only point to instances of that but also to instances of subclasses of the same. One can even create subclass objects using a reference variable typed with respect to a super class.

❖Polymorphism through inheritance can be achieved by playing with static and dynamic type of a reference variable.

❖Instances of a subclass may be used through the super class's interface. When this is done, a client can't access all components defined in the subclass, only those inherited from the respective super class.

Inheritance – Object References



OBJECT

C1

PUB: **a1 m1**

PROT: pro_a1 pro_m1

PRIV: f1 pri_m1

C2 INHERITING FROM C1

PUB: **a1** a2 **m1** m2

PROT: **pro_a1** pro_a2 **pro_m1** pro_m2

PRIV: f1 pri_m1

C3 INHERITING FROM C2

PUB: **a1 a2** a3 **m1 m2** m3

PROT: **pro_a1 pro_a2** pro_a3 **pro_m1 pro_m2** pro_m3

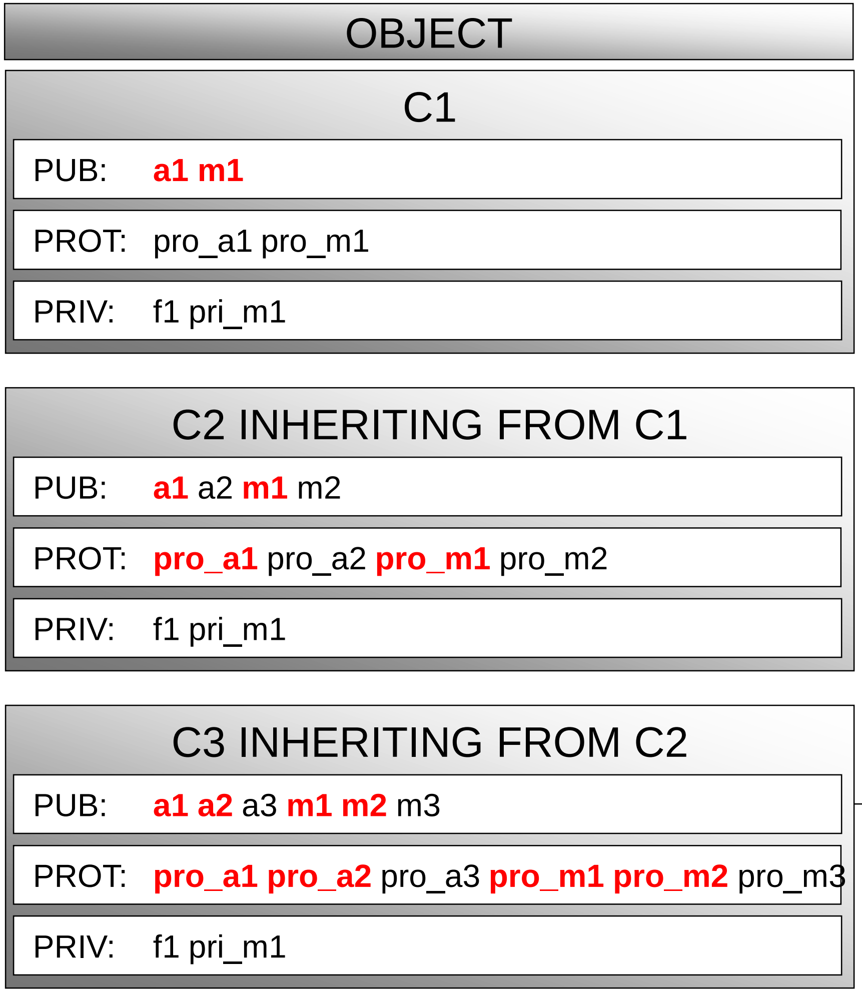
PRIV: f1 pri_m1

External Interface

```
DATA: oref1 TYPE REF TO C1,
      oref3 TYPE REF TO C3.
...
CREATE OBJECT oref3.
oref1 = oref3.
```

Reference variables declared with reference to a superclass (including an abstract class) can point to an object of a subclass of this superclass, but only access the components of that object that are known to the superclass.

Inheritance – Object References (3)



External Interface

```
DATA: oref1 TYPE REF TO C1.
...
CREATE OBJECT oref1 TYPE C3.
```

a1

m1

The CREATE OBJECT statement has a TYPE addition with which you can specify the class from which the object is to be created. This becomes the dynamic type of the reference variable. Its static type, which never changes, is the type with which it was declared.

Topics to cover

- ❑ Different approaches to Programming
- ❑ Class and objects
- ❑ Methods
- ❑ Constructor
- ❑ Inheritance
- ❑Interface
- ❑Friendship between Classes
- ❑ Events
- ❑ Class-Based Exceptions

Interface

❖Independent structures containing definition of its own components.

❖Incorporated in the public section of a class definition and the methods of the interface are implemented in the implementation section of that class.

✓Extends the scope of a class by adding their own components to its public section.

✓Allows users to address different classes via a universal point of contact.

✓Interfaces, along with inheritance, provide one of the pillars of polymorphism, since they allow a single method within an interface to behave differently in different classes.

Defining Interfaces

```
report ysubdel .
interface i1.
 data    : num type i .
 methods : meth1.
endinterface.

 class c1 definition.
  public section.
  methods : meth1.
  interfaces : i1.
 endclass.

 class c1 implementation.
  method : meth1.
   write:/5 'I am meth1 in c1'.
  endmethod.

  method i1~meth1.
   write:/5 'I am meth1 from i1'.
  endmethod.
 endclass.

 start-of-selection.
  data : oref type ref to c1.  create object oref.
  write:/5 oref->i1~num.
  call method oref->meth1.
  call method oref->i1~meth1.
```

❖Can be declared globally or locally within a program.

❖Locally declared in the global portion of a program using:-

> **❖INTERFACE <intf>.**
>
> **❖...**
>
> **❖ENDINTERFACE.**

❖The definition contains the declaration for all components (attributes, methods, events) of the interface.

❖Interfaces are included in the public section of a class.

❖ Interfaces do not have an implementation part, since their methods are implemented in the class that implements the interface.

Defining Interfaces…. A few additions

(1) INTERFACE <intf> DEFERRED.

Usually, an interface is first defined and then used.

But, if there is a need to use an interface in a context before it is defined - the interface must be made known using the **INTERFACE intf DEFERRED** variant. There is no ENDINTERFACE statement for this variant.

(2) INTERFACE <intf> LOAD.

When an interface is used for the first time in a program - particularly if the interface is used for the first time to type reference variables , the compiler automatically loads the description of a global interface from the class library . However, if the first use of a global interface is accessing one of its static component, or when an event handler method is defined with respect to an event in the global interface, the interface description must be loaded explicitly using the INTERFACE intf LOAD (for technical reasons). In such cases, there is no associated ENDINTERFACE statement.

Defining Interfaces…. Additions for Implementation in Classes

One cannot use the VALUE addition with the DATA statement in interface. Nor can one specify the method types while defining them. Everything is to be done with the INTERFACE statement.

Addition	Effect
... ABSTRACT METHODS meth_1 ... meth_n	Assigns the property **ABSTRACT** to the specified instance methods of the interface.
... FINAL METHODS meth_1 ... meth_n	Assigns the property **FINAL** to the specified instance methods of the interface.
... ALL METHODS ABSTRACT	Assigns the property **ABSTRACT** to all instance methods of the interface.
... ALL METHODS FINAL	Assigns the property **FINAL** to all instance methods of the interface.
... DATA VALUES attr_1 = val_1 ... attr_n = val_n	Assigns initial values to the attributes specified (**DATA**, **CLASS-DATA**). The values of constants cannot be changed. The specifications **val_n** have the same meaning as the **VALUE** specification in the DATA statement.

Component of Interfaces

The interface **components** are declared in the INTERFACE…ENDINTERFACE block.

Instance Components		**Static Components**	
DATA	for instance attributes	**CLASS-DATA**	for static attributes
METHODS	for instance methods	**CLASS-METHODS**	for static methods
EVENTS	for instance events	**CLASS-EVENTS**	for static events
TYPES	for internal class types	**CONSTANTS**	For constants

Other Interface Components

INTERFACES for nesting interfaces

ALIASES for aliases for components of interfaces

One cannot use the VALUE addition with the DATA statement in interface. The INTERFACES statement is used to nest and compound interfaces, not to implement them.

Implementing Interfaces

Interfaces do not have instances. Rather, they are implemented by classes. For that, interfaces are included in the public section of a class definition using the following syntax:-

```
INTERFACES <intf>.
```

The class must implement the methods of all interfaces implemented in it.

```
METHOD <intf~imeth>.

...

ENDMETHOD.
```

Interfaces can be implemented by different classes and the methods of the interface can be implemented differently in each class.

Interface Reference

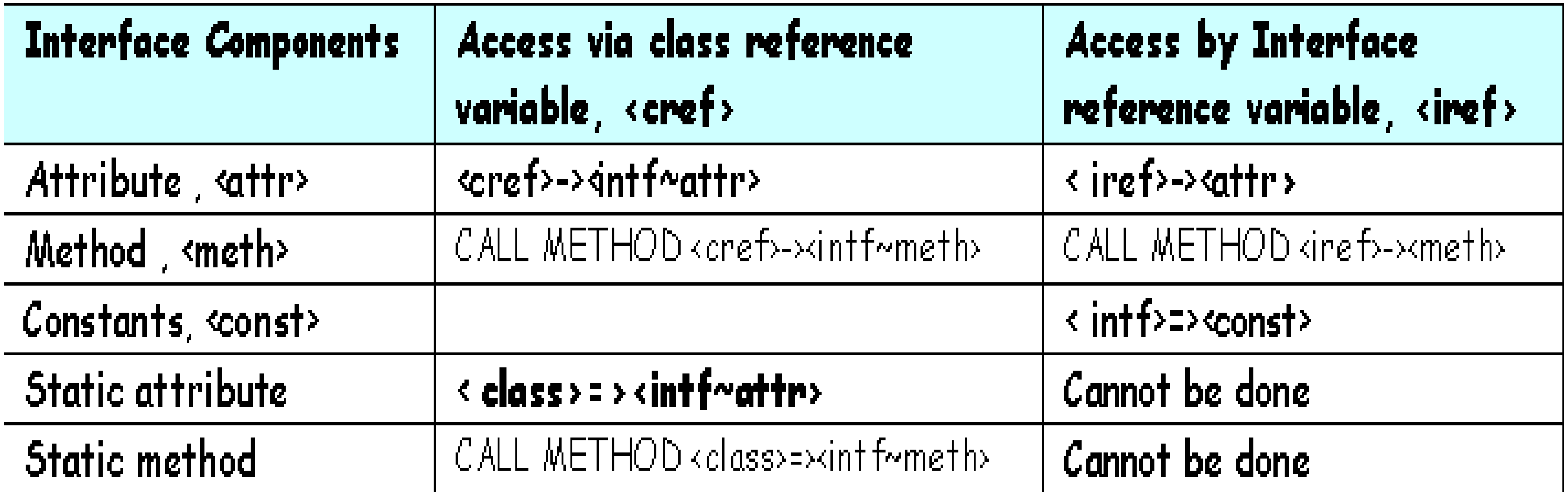
Like classes , reference variables can be created with respect to an interface This kind of reference variable can contain references to objects of classes that implement the corresponding interface.

To define an interface reference, the addition TYPE REF TO <intf> in the TYPES or DATA statement is used. <intf> must be an interface that has been declared to the program before the actual reference declaration occurs.

Addressing Objects using Interface Reference

If a class <class> implements an interface <intf>, the interface reference variable <iref> can be assigned the class reference variable <cref> to make the interface reference in <iref> point to the same object as the class reference in <cref>. The syntax for that is :-

<iref> = <cref>

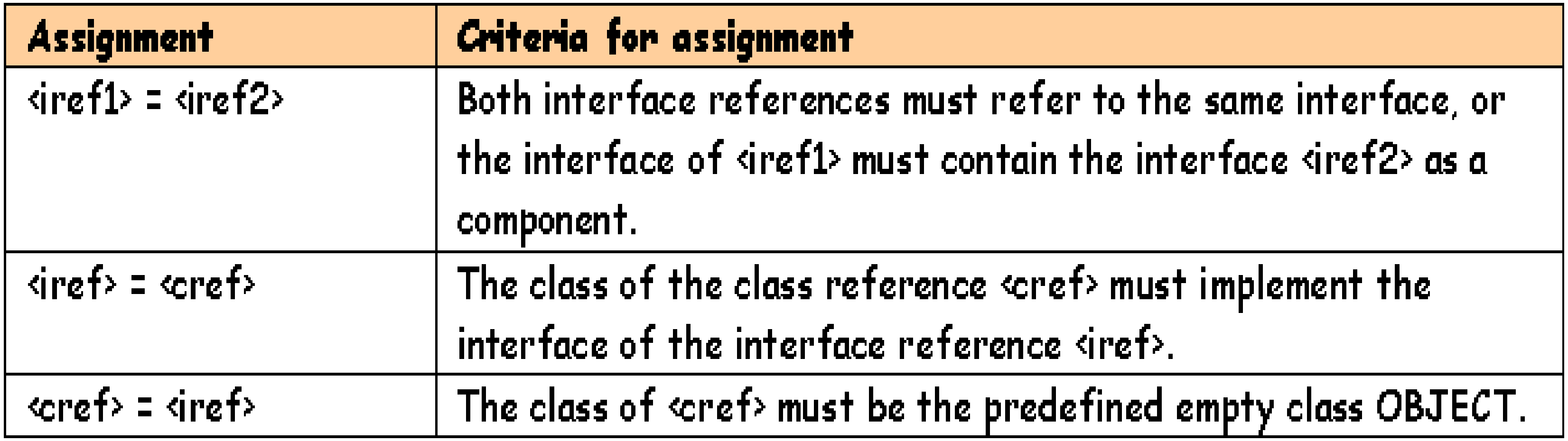


Interface Components	Access via class reference variable, <cref>	Access by Interface reference variable, <iref>
Attribute , <attr>	<cref>-><intf~attr>	< iref>-><attr >
Method , <meth>	CALL METHOD <cref>-><intf~meth>	CALL METHOD <iref>-><meth>
Constants, <const>		< intf>=><const>
Static attribute	< class>= ><intf~attr>	Cannot be done
Static method	CALL METHOD <class>=><intf~meth>	Cannot be done

Assignment using Interface References

One can assign interface references to other interface/class reference variables, using MOVE statement or the assignment operator (=) . However, there are certain restrictions while doing so. When one does this in a program, system must be able to recognize in the syntax check whether an assignment is possible.

Suppose we have a class reference <cref> and interface references <iref>, <iref1>, and <iref2>. The following assignments with interface references can be checked statically:



Assignment	Criteria for assignment
<iref1> = <iref2>	Both interface references must refer to the same interface, or the interface of <iref1> must contain the interface <iref2> as a component.
<iref> = <cref>	The class of the class reference <cref> must implement the interface of the interface reference <iref>.
<cref> = <iref>	The class of <cref> must be the predefined empty class OBJECT.

Assignment using Interface References…check at Runtime

Besides static check, dynamic check to ensure consistency of assignment of interface references.

For that, the casting operator (?= or MOVE …? TO) can be used. The system then performs at **runtime** .If the assignment is possible, the system makes it, otherwise, the catchable runtime error MOVE_CAST_ERROR occurs.

Interfaces – Defining and Implementing Compound Interfaces

```
INTERFACE i1.
   METHODS meth.
ENDINTERFACE.

INTERFACE i2.
   METHODS meth.
   INTERFACES i1.
ENDINTERFACE.

INTERFACE i3.
   METHODS meth.
   INTERFACES: i1, i2.
ENDINTERFACE.

INTERFACE i4.
   INTERFACES  i3.
ENDINTERFACE.
```

```
CLASS c1 DEFINITION.
   PUBLIC SECTION.
      INTERFACES: i4.
ENDCLASS.

CLASS c1 IMPLEMENTATION.
   METHOD i1~meth.
     …
   ENDMETHOD.
   METHOD i2~meth.
     …
   ENDMETHOD.
   METHOD i3~meth.
     …
   ENDMETHOD.
ENDCLASS.
```

- You can compose a new interface from several existing ones. Such an interface is called a **compound interface**; an interface which is contained in another interface is called a **component interface**
- There is no component hierarchy: All component interfaces are on the same level, so it is not possible to chain names such as `i3~i2~i1`
  - A compound interface contains each component interface only once
  - When a class implements interfaces, each component interface is implemented only once

Interfaces – Alias Names

```
INTERFACE i1.
   METHODS meth.
ENDINTERFACE.

INTERFACE i2.
   INTERFACES i1.
   ALIASES m1 FOR i1~meth.
ENDINTERFACE.

INTERFACE i3.
   INTERFACES i1, i2.
   ALIASES m1 FOR i2~m1.
ENDINTERFACE.
```

```
CLASS c1 DEFINITION.
   PUBLIC SECTION.
      INTERFACES: i1.
      ALIASES m1 for
i1~meth.
ENDCLASS.

CLASS c1 IMPLEMENTATION.
   METHOD i1~meth.
      …
   ENDMETHOD.
ENDCLASS.

DATA: oref TYPE REF TO c1.
CREATE OBJECT oref.
CALL METHOD oref->m1.
```

- The full name of a component which an Interface adds to a class or another interface is `intf~comp`. Alias names can be substituted for this name when
  - defining compound interfaces, or
  - declaring Interfaces in a class
- Since the chain name `i3~i2~meth` is not allowed, the alias name `m1` in `i2` enables `i3`'s own ALIASES statement to address component `meth` in `c1`
- The class implementation must still refer to the full name, but the user can use the alias. This permits
  - Code that is easier to write and read
  - A class publishing its interface components as class-specific components (`m1` rather than `i1`meth`)
  - Replacing class-specific components (e.g., `meth`) with interfaces (e.g., ALIASES `meth` for `m1~intf_method`), without affecting existing users of the component

Topics to cover

- ❑ Different approaches to Programming
- ❑ Class and objects
- ❑ Methods
- ❑ Constructor
- ❑ Inheritance
- ❑ Interface
- ❑ Friendship between Classes
- ❑ Events
- ❑ Class-Based Exceptions

Friends

In classes, there is normally a strict division between outside (**PUBLIC**) and inside (**PROTECTED** or **PRIVATE**). A user can only access the public components of a class. A subclass have access to public and protected sections of its superclass.

In rare cases, classes have to work together so closely that they need access to one anothers' protected or private components. In order to prevent these components being available to all users, there is the concept of friendship between classes.

Friends…continued

❖A class can declare other classes and interfaces (and hence all classes that implement this interface) as friends by including **FRIENDS** additions to the CLASS ... DEFINITION statement which lists all the classes and interfaces that are granted friendship.

❖These friends gain access to the protected and private components of the class granting the friendship and can always create instances of this class, independently from the CLASS statement's **CREATE** addition.

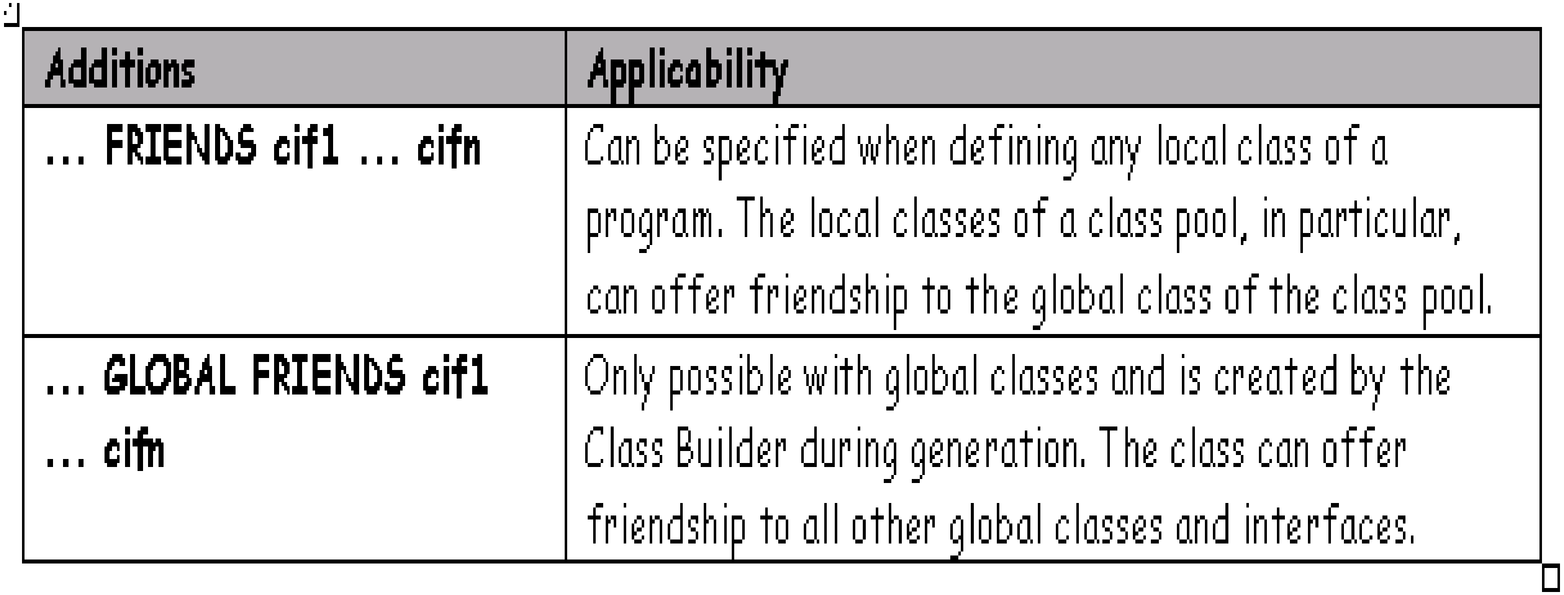
❖In principle, granting of friendship is one-sided: A class granting a friendship is **not** automatically a friend of its friends. If the class granting the friendship wants to access the private components of a friend, then the latter has to explicitly grant friendship to the former.

❖However, the granting of friendship is not passed on: A friend of a superclass is not automatically a friend of its subclasses.

❖But, subclasses of friend class also become friend of the current class too.

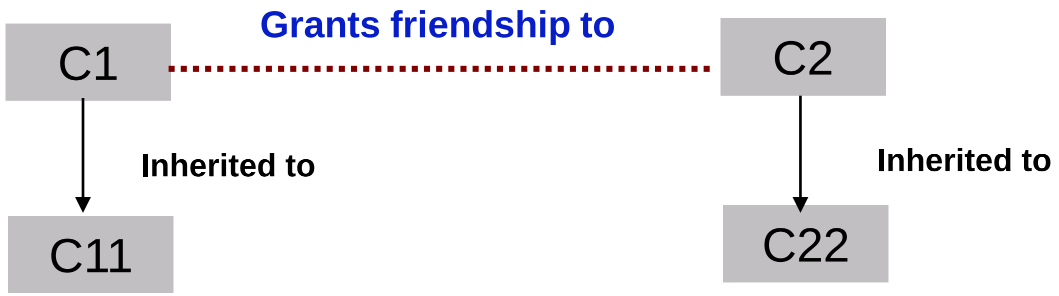
Various Additions to the Friends

The **CLASS … DEFINITION** statement has mainly two different **FRIENDS** additions to offer friendship to other classes/interfaces:



Additions	Applicability
… FRIENDS cif1 … cifn	Can be specified when defining any local class of a program. The local classes of a class pool, in particular, can offer friendship to the global class of the class pool.
… GLOBAL FRIENDS cif1 … cifn	Only possible with global classes and is created by the Class Builder during generation. The class can offer friendship to all other global classes and interfaces.

Friends…continued



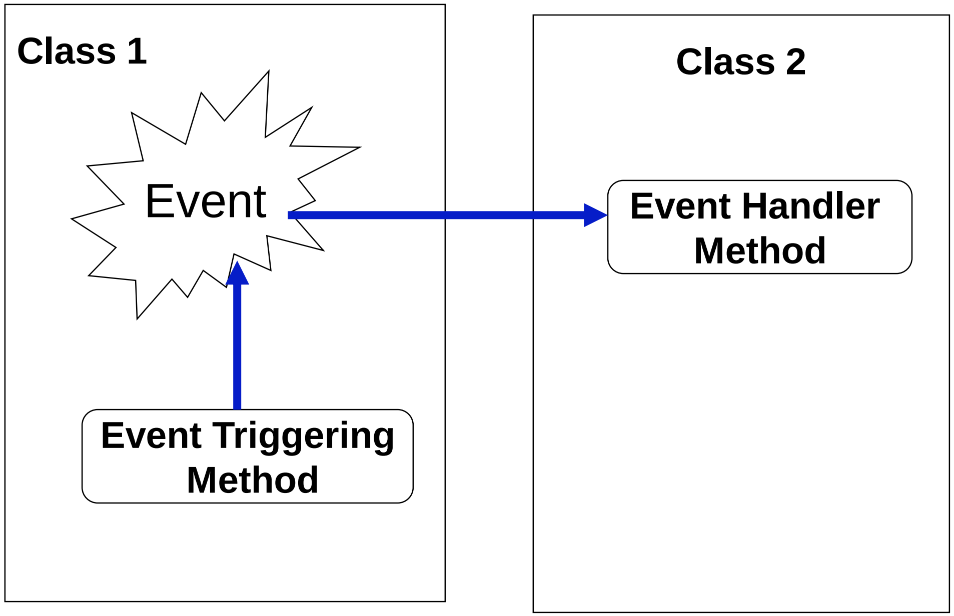
Statement	Yes/ No
C2 can access all attributes/methods of C1 from any section.	Y
C2 can create instances of C1, irrespective of its CREATE addition	Y
C1 can access all attributes/methods of C2 from any section.	N
C1 can create instances of C2, irrespective of its CREATE addition	N
C2 can access all attributes/methods of C11(subclass of C1) from any section.	N
C2 can create instances of C11, irrespective of its CREATE addition	N
C22 (subclass of C2) can access all attributes/methods of C1 from any section.	Y
C22(subclass of C2) can create instances of C1, irrespective of its CREATE addition	Y

Topics to cover

- ❑ Different approaches to Programming
- ❑ Class and objects
- ❑ Methods
- ❑ Constructor
- ❑ Inheritance
- ❑ Interface
- ❑ Friendship between Classes
- ❑ Events
- ❑ Class-Based Exceptions

Events

Raising events in Object Oriented Programming is a mechanism by which methods of one class can call methods of same/other classes.



List of To-do's

1.Declare an event in a class.

2. Declare a triggering method and implement it In the same class.

3. Declare an event handler Method in same/other class.

4.Register the event handler Method with the event at Runtime.

5. Call the triggering method.

Declaring events

Events are declared in the declaration part of a class or in an interface.

The syntax is :-

EVENTS <evt> [EXPORTING... VALUE(<e$_i$>) TYPE type [OPTIONAL]..]

To declare static events, use the following statement:

CLASS-EVENTS <evt>...

(Both statements have the same syntax).

One can send values to the EXPORTING parameters of an event which are eventually passed to the event handler method. The parameters are always passed by value. It is not mandatory that the event will pass all the inputs fed to it to the registered event handler method.

Instance events always contain the implicit parameter SENDER, which has the type of a reference to the type or the interface in which the event is declared.

Triggering Events

An instance event in a class can be triggered by any method in the class. Static events can be triggered by any static method. The following syntax should be used inside the triggering event in the implementation part of the class to trigger the event residing in the same class:-

RAISE EVENT <evt> EXPORTING... <e$_i$> = <f $_i$>...

For each formal parameter **<e i >** that is not defined as optional, one must pass a corresponding actual parameter **<f i >** in the EXPORTING addition.

Triggering events have to be in the **PUBLIC** section if the method is to be called externally.

Declaring Event Handler Methods

Any class can contain event handler methods for events from other classes. Even, one can also define event handler methods in the same class as the event itself. To declare an instance event handler method in the PUBLIC section of a class,   the following statement should be followed: -

```
METHODS <meth> FOR EVENT <evt> OF <cif> IMPORTING.. <ei>..
```

For a static method, use CLASS-METHODS instead of METHODS. <evt> is an event declared in the class or interface <cif>.

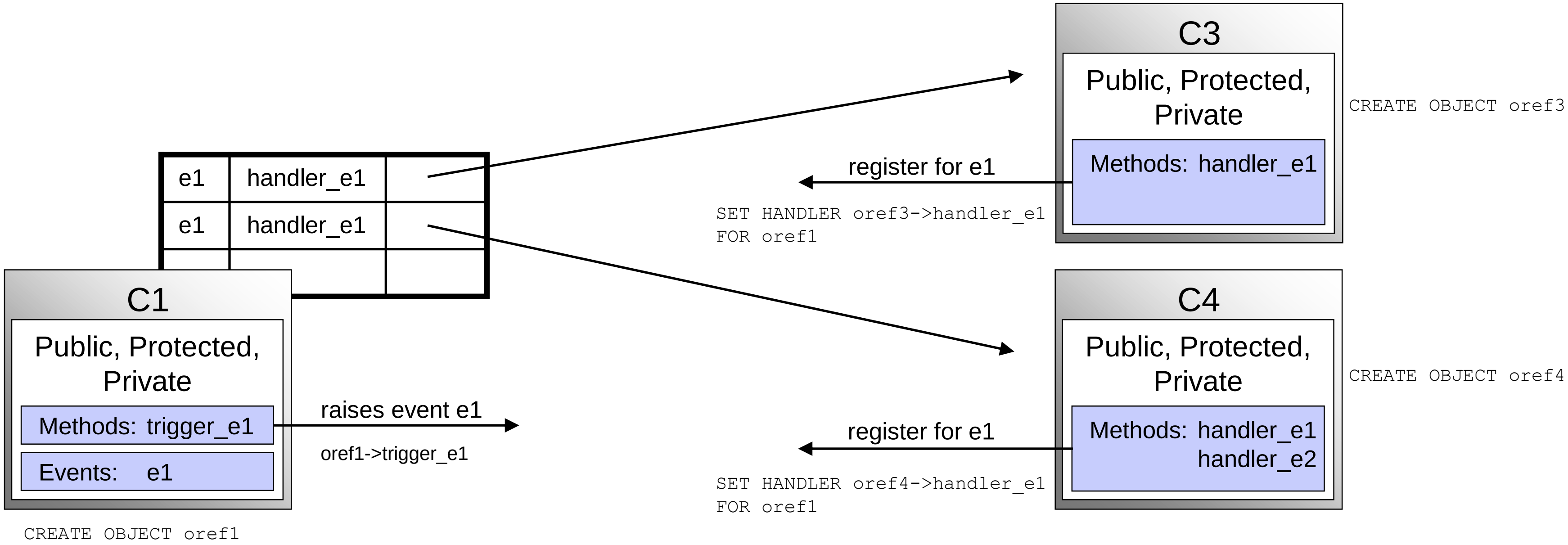
The interface of an event handler method may only contain formal parameters defined in the declaration of the event <evt>. The attributes of the parameter are also adopted by the event. The event handler method does not have to use all of the parameters passed in the RAISE EVENT statement.

Events – Registering

```
START-OF-SELECTION.
   * c1 declares and raises event e1
   * c2 handles event e1
   CREATE OBJECT  ref_to_c1,
                  ref_to_c2.

   SET HANDLER ref_to_c2->event_handler
               FOR ref_to_c1.

   CALL METHOD:
           ref_to_c1->trigger_event.
```

- So far we have statically declared an event and implemented a trigger and a corresponding handler
- To make the handler actually work in runtime, though, it must be registered (it must "subscribe" to the event). To do this, use the syntax `SET HANDLER ref_handler FOR ref_sender [ACTIVATION act]`
  - In the above syntax, `ref_sender` refers to a single object. You can mass register for all objects that can trigger the event using … `FOR ALL INSTANCES` … instead
  - You can undo the registration by using the `SET HANDLER` statement with `ACTIVATION` set to ' ' (by default, it is set to 'X')

Events – Runtime Environment



- Each object that can trigger events has an invisible handler table, where the entries occur as a result of SET HANDLER statements
- The RAISE EVENT statement in the triggering method interrupts the execution of the method until the runtime finishes cascading through each handler registered in the handler table (an event handler method can itself raise an event). The initial triggering method then resumes execution
- Note: Should the program delete, say, reference variable `oref3`, the object it references is still pointed to by `oref1's` handler table and therefore will not be deleted until the object pointed to by *oref1* is itself deleted

Events with Parameters

- Events can have export parameters, which it passes to its event handler method.
- The triggering method must pass values for all the exporting parameters of the event while raising the event using **RAISE EVENT** statement.
- The interface of an event handler method consists of a list of **IMPORTING** parameters, whose names are identical with those in the EXPORTING list and which are automatically created from the interface of the event.
- Each handler method can however specify which event parameters it wants to handle and which it does not.

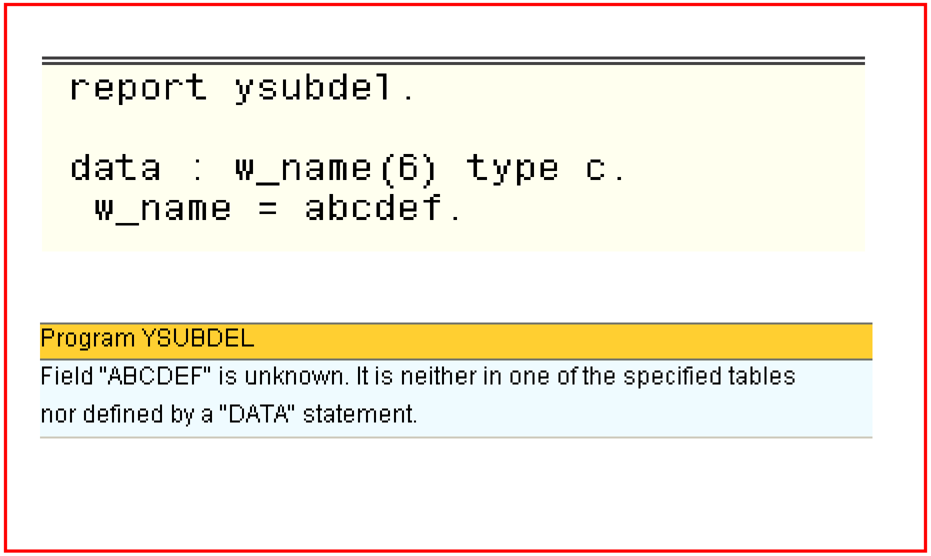
Topics to cover

- ❑ Different approaches to Programming
- ❑ Class and objects
- ❑ Methods
- ❑ Constructor
- ❑ Inheritance
- ❑ Interface
- ❑ Friendship between Classes
- ❑ Events
- ❑ Class-Based Exceptions

What is an Exception?

An exception is a situation that occurs during the execution of an ABAP program, which renders a normal program continuation pointless.

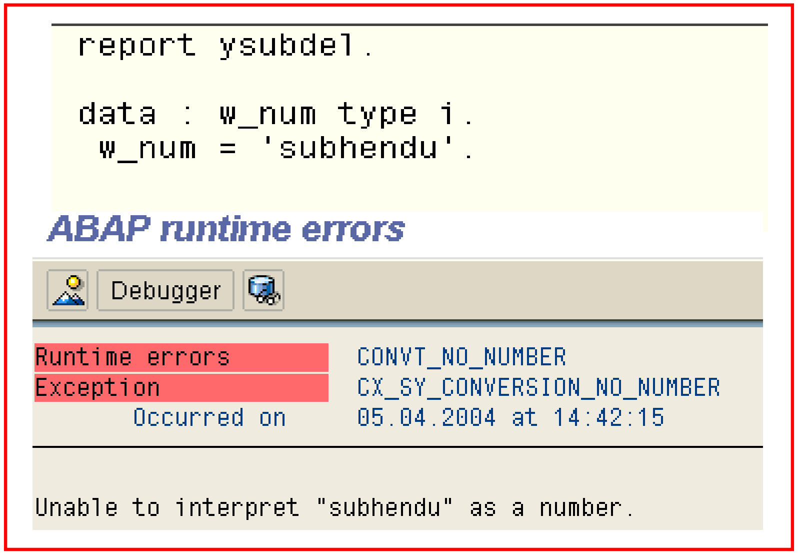
Exceptions can be detected at the time of program compilation or at runtime.If the exception detected at runtime is not handled properly by the program itself, we get a short dump and the execution terminates.



```
 report ysubdel.

 data : w_name(6) type c.
  w_name = abcdef.
```

Program YSUBDEL

Field "ABCDEF" is unknown. It is neither in one of the specified tables nor defined by a "DATA" statement.



```
 report ysubdel.

 data : w_num type i.
  w_num = 'subhendu'.
```

ABAP runtime errors

```
Runtime errors          CONVT_NO_NUMBER
Exception               CX_SY_CONVERSION_NO_NUMBER
       Occurred on      05.04.2004 at 14:42:15

Unable to interpret "subhendu" as a number.
```

Classification of Exceptions

Exceptions of various kinds can be broadly classified as :-

- Exceptions that can be handled.
- Exceptions that cannot be handled.

Exceptions that can be handled indicate error situations in the runtime environment or in the ABAP program, in the case of which the program execution can be continued - by handling the exception in the ABAP program - without the system reaching a critical condition. **If such a situation is not handled a runtime error will occur.**

Exceptions that cannot be handled indicate critical error situations in the runtime environment, which cannot be handled with/by ABAP means and always cause a runtime error. Database space problem can be an example of such category.

Traditional Ways of Catching Runtime Exceptions

Areas	Brief Overview
In ABAP	**catch system-exceptions** <exception_name> = <val>. . . . . **Endcatch.** **If sy-subrc** = <val> . < exception handling statements> **Endif.**
In function module	Creating exceptions for function module, raising them at appropriate points in the FM , assigning different sy-subrc values for each exceptions at the time of the FM call and later dealing with them.
In Methods	Creating different exceptions at the time of declaring methods, raising those exceptions within the method, assigning different sy-subrc values at the time of method call and later dealing with those values.

What is Class-based exception handling?

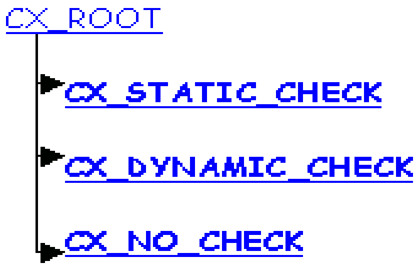
- In Class-based exceptions handling approach, exceptions are generally represented by objects of exception classes. There are pre-defined exception classes for error situations in the runtime environment .
- Users can also define own exception classes globally/locally, if required and can raise them using **RAISE EXCEPTION** statement.
- The runtime environment only causes exceptions that are based on pre-defined classes, while in ABAP programs one can use raise pre-defined as well as user-specific exception classes.
- Class-based exceptions are handled using the control structure **TRY ... ENDTRY.**
- Class-based exceptions in procedures can be propagated to the caller in the definition of the interface using the **RAISING** addition, if the exception is not to be handled in the procedure.

TRY…CATCH…ENDTRY

Class-based exceptions are handled using TRY…CATCH…ENDTRY block.

```
TRY.
< code to be checked for exception>
CATCH  cx1 ….cxn … [ into ref].
< exception handling code>.
ENDTRY.
```

```
REPORT  YSUBCLASS_EXCEPTION.

 DATA: i TYPE i VALUE 1.

START-OF-SELECTION.
        TRY.
        i = i / 0.
        CATCH cx_sy_zerodivide.
         write:/5 'Divide by zero caught'.
      ENDTRY.
```



Class-Based Exceptions – SAP Exception Classe

- **CX_STATIC_CHECK**:
  - For exceptions that **have to be declared.** This type should be chosen if you want to make sure that this exception is always dealt with and if a local exception handler has a chance to do something useful in an exception situation
  - Corresponding exceptions must either be handled or forwarded explicitly with the **RAISING** addition and this is **checked at syntax check**
- **CX_DYNAMIC_CHECK**:
  - For exceptions that **do not have to be declared**
  - Exceptions must be handled or explicitly forwarded with the **RAISING** addition though this is **not checked at syntax check.** Exceptions of this type are checked at runtime only
  - Useful for potential error situations that do not have to be handled, since the program logic can more or less exclude them. Example: cx_sy_zerodivide
  - **Most of the CX_SY_** exceptions inherit from this class
- **CX_NO_CHECK:**
  - For exceptions that **must not be declared** (i.e. resource bottlenecks)
  - Can be handled but not forwarded with RAISING. Otherwise will be propagated through call chain **automatically**
  - **Not checked by syntax check** or **runtime processing**

SAP Exception Classes

SAP provided exception-classes are derived from the specific class CX_ROOT and have the prefix CX_.

Exception classes are normal classes with one limitation:-

Apart from the constructor, no methods can be defined for them. However, CX_ROOT has some pre-defined methods available, which can then be inherited by all exception classes.

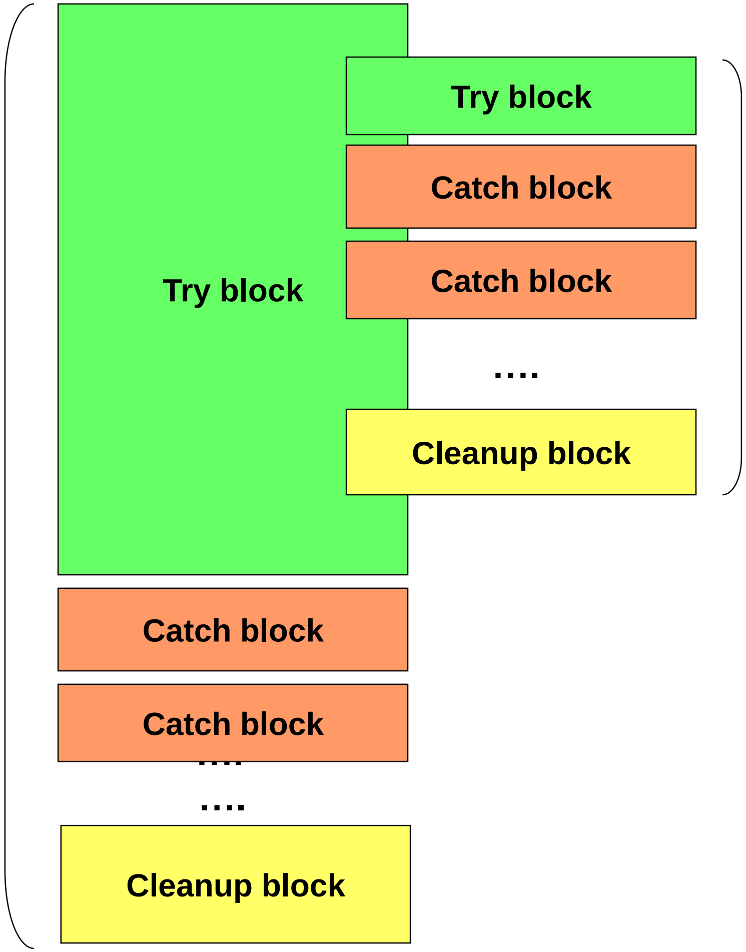
Component Name	(M)ethod/(A)ttribute	Description
GET_TEXT	M	Returns a text description of the exception
GET_SOURCE_POSITION	M	Returns the point at which the exception occurred
TEXTID	A	Used to define different texts for exceptions of a particular exception class. Affects the result of the method GET_TEXT.

SAP Exception Classes

Component Name	(M)ethod/(A)ttribute	Description
PREVIOUS	A	If one exception is mapped to another, this attribute stores the original exception, which allows the system to build a chain of exceptions.
KERNEL_ERRID	A	Contains the name of the appropriate runtime error if the exception was triggered from the kernel. If the exception was raised using a RAISE EXCEPTION, this attribute is initial.
TEXTID	A	Used to define different texts for exceptions of a particular exception class. Affects the result of the method GET_TEXT.

Nested Try…Catch…Endtry Blocks

```
TRY.
    TRY.

          CATCH cx_class INTO oref

          CATCH cx_class INTO oref

          …

          CLEANUP.

    ENDTRY.

    CATCH cx_class INTO oref.

    CATCH cx_class INTO oref.

    …
    CLEANUP.

ENDTRY.
```



CLEANUP

```
Report ysubdel.
data : w_num type i.
 try.
   try .
    w_num = 5 / 0 .
    cleanup.
     write:/5 ‘In cleanup’.
   endtry .
  catch cx_sy_zerodivide.
   write:/5 ‘Div. By zero!’.
 endtry.
```

```
 In cleanup

Div. by zero!
```

- Used within a TRY…ENDTRY BLOCK , after all CATCH statements.
- Each TRY block can contain maximum of one CLEANUP area.
- Used to release the external resources when exception detected in a TRY block is not handled within the block , but is caught further up in the call hierarchy.
- Possible only in cases of nested TRY blocks.

Creating Local Exception Class in a program

To create a local exception class in a program and use it, follow the steps outlined below.

Step 1 :- Create a subclass from global exception class in your program.

```
REPORT  YSUBCLASS_EXCEPTION_3.
 CLASS CX_SOME_EXCEPTION DEFINITION   INHERITING FROM
CX_STATIC_CHECK.
public section.
 methods : meth1.
ENDCLASS.
```

Creating Local Exception Class in a program

Step 2 :- Implement methods of the subclass which will raise exception

```
CLASS CX_SOME_EXCEPTION IMPLEMENTATION.
method : meth1.
 write:/5 'I am a method in exception'.
endmethod.
ENDCLASS.
```

Step 3 :- Define another class which will call the exception class.

```
CLASS SOME_CLASS DEFINITION.
  PUBLIC SECTION.
    METHODS: m1 raising cx_some_exception .
ENDCLASS.
```

Creating Local Exception Class in a program

Step 4 :- Implement the method of the other class which will raise exception of the locally declared exception class.

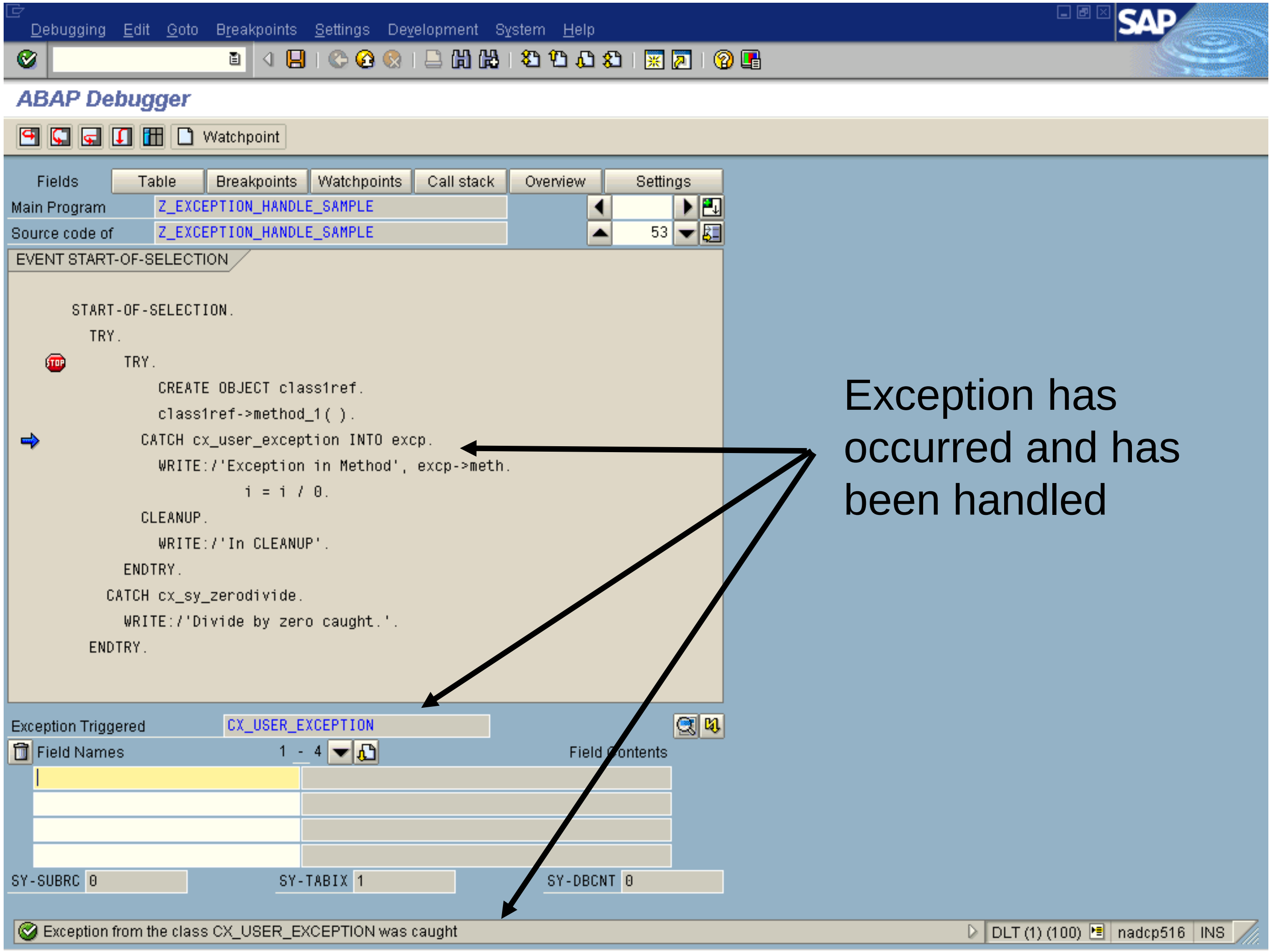
```
CLASS SOME_CLASS IMPLEMENTATION.
  METHOD m1.
   RAISE EXCEPTION TYPE CX_SOME_EXCEPTION.
  ENDMETHOD.
ENDCLASS.
```

Creating Local Exception Class in a program

Step 5 :- Create an object of the other class and call its method which will raise the exception

```
DATA:  c1 TYPE REF TO SOME_CLASS.
 START-OF-SELECTION.
 TRY.
   CREATE OBJECT c1.
   c1->m1( ).
 CATCH CX_some_exception.
   write:/5 'Exception caught'.
 ENDTRY.
```

Class-Based Exceptions – Debug Mode



ABAP Debugger

Main Program: Z_EXCEPTION_HANDLE_SAMPLE

Source code of: Z_EXCEPTION_HANDLE_SAMPLE   53

EVENT START-OF-SELECTION

```
START-OF-SELECTION.
  TRY.
      TRY.
          CREATE OBJECT class1ref.
          class1ref->method_1( ).
        CATCH cx_user_exception INTO excp.
          WRITE:/'Exception in Method', excp->meth.
                    i = i / 0.
        CLEANUP.
          WRITE:/'In CLEANUP'.
      ENDTRY.
    CATCH cx_sy_zerodivide.
      WRITE:/'Divide by zero caught.'.
  ENDTRY.
```

Exception Triggered: CX_USER_EXCEPTION

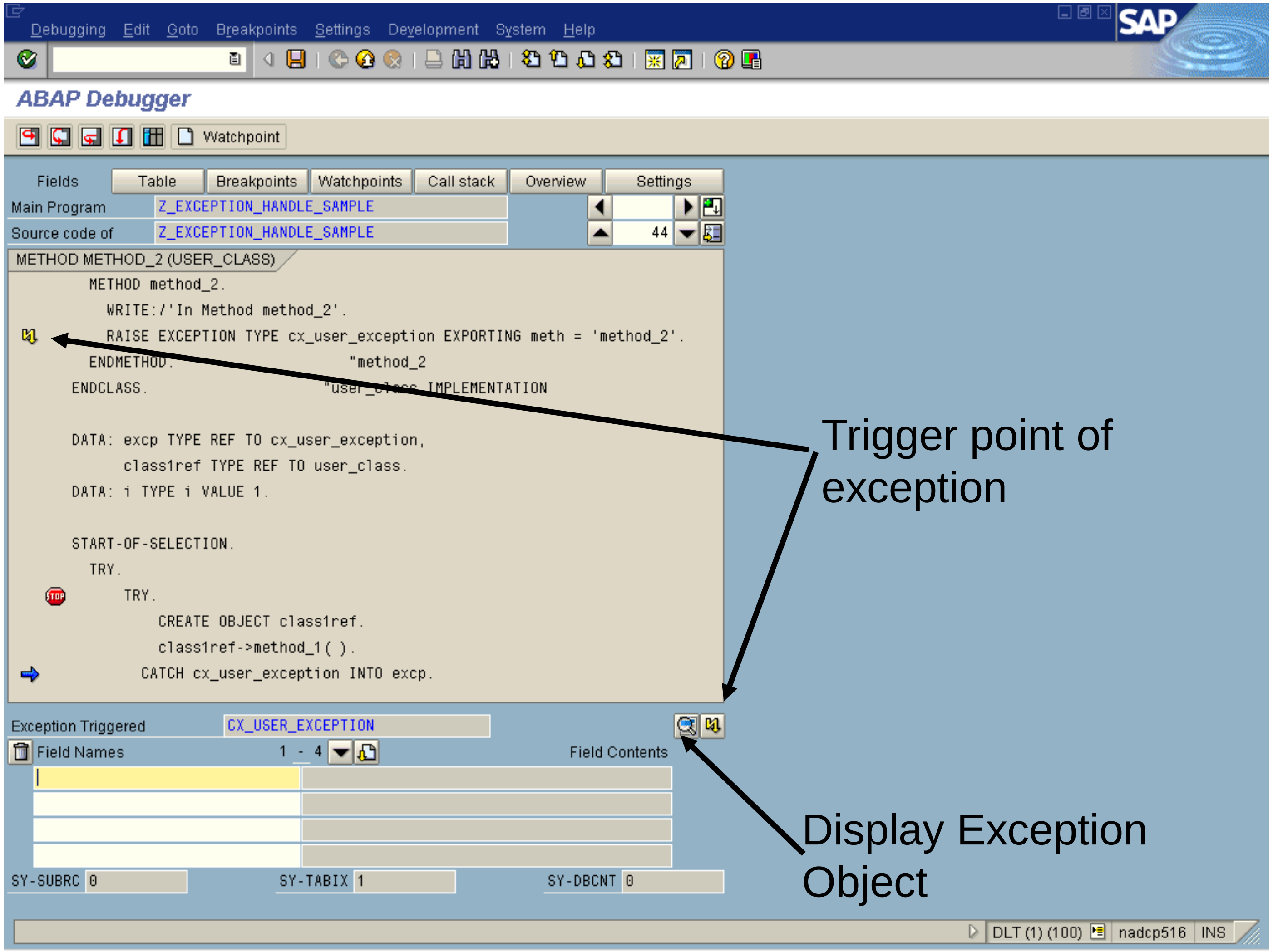
Field Names   1 - 4   Field Contents

SY-SUBRC 0   SY-TABIX 1   SY-DBCNT 0

Exception from the class CX_USER_EXCEPTION was caught   DLT (1) (100)   nadcp516   INS

Exception has occurred and has been handled

Class-Based Exceptions – Debug Mode



ABAP Debugger

Main Program Z_EXCEPTION_HANDLE_SAMPLE

Source code of Z_EXCEPTION_HANDLE_SAMPLE 44

METHOD METHOD_2 (USER_CLASS)

```
      METHOD method_2.
        WRITE:/'In Method method_2'.
        RAISE EXCEPTION TYPE cx_user_exception EXPORTING meth = 'method_2'.
      ENDMETHOD.                    "method_2
    ENDCLASS.                    "user_class IMPLEMENTATION

    DATA: excp TYPE REF TO cx_user_exception,
          class1ref TYPE REF TO user_class.
    DATA: i TYPE i VALUE 1.

    START-OF-SELECTION.
      TRY.
          TRY.
              CREATE OBJECT class1ref.
              class1ref->method_1( ).
            CATCH cx_user_exception INTO excp.
```

Exception Triggered CX_USER_EXCEPTION

Field Names 1 - 4

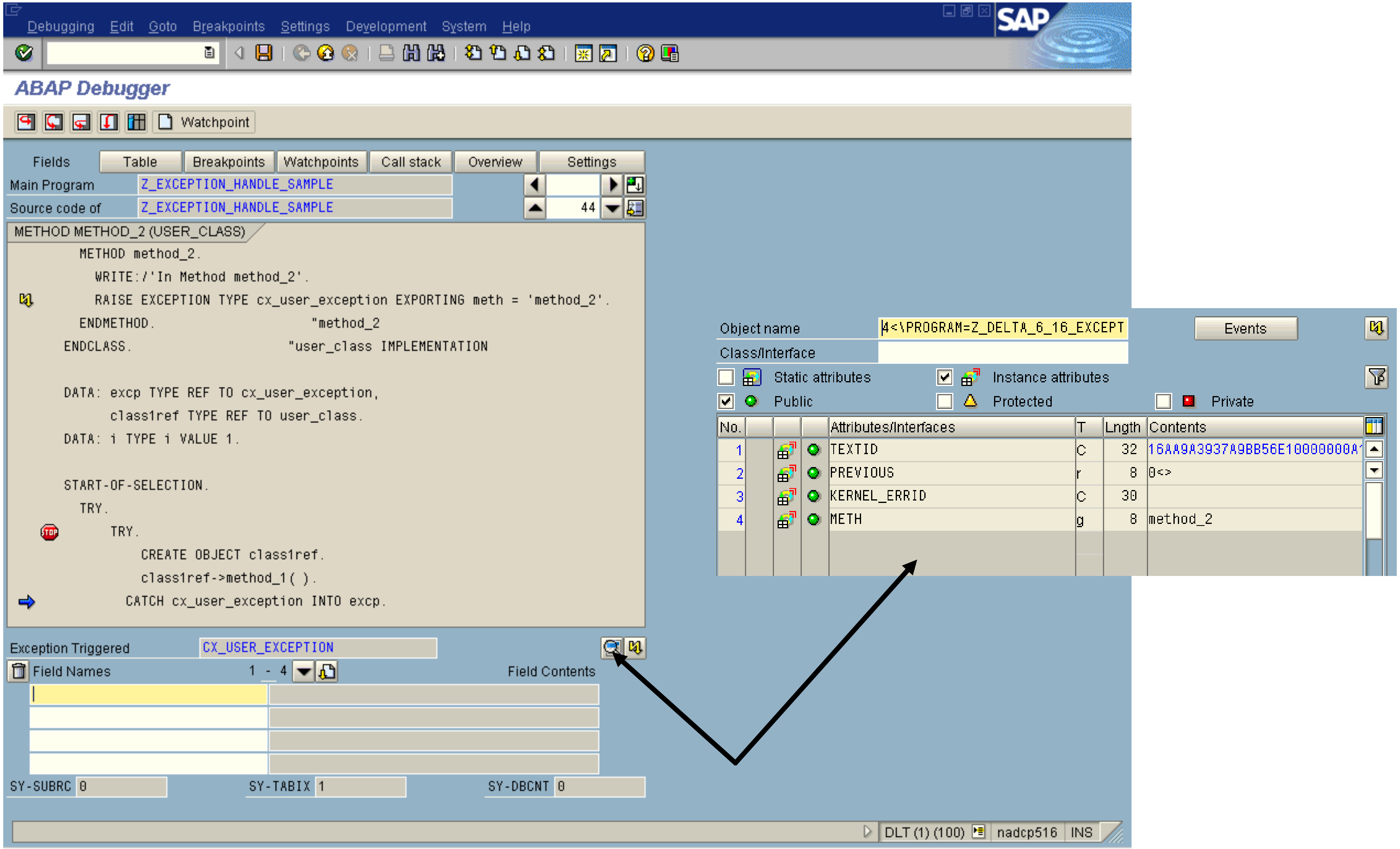
SY-SUBRC 0 SY-TABIX 1 SY-DBCNT 0

DLT (1) (100) nadcp516 INS

Trigger point of exception

Display Exception Object

Class-Based Exceptions – Debug Mode



Main Program: Z_EXCEPTION_HANDLE_SAMPLE

Source code of: Z_EXCEPTION_HANDLE_SAMPLE   44

METHOD METHOD_2 (USER_CLASS)

```
    METHOD method_2.
      WRITE:/'In Method method_2'.
      RAISE EXCEPTION TYPE cx_user_exception EXPORTING meth = 'method_2'.
    ENDMETHOD.                    "method_2
  ENDCLASS.                    "user_class IMPLEMENTATION

  DATA: excp TYPE REF TO cx_user_exception,
        class1ref TYPE REF TO user_class.
  DATA: i TYPE i VALUE 1.

  START-OF-SELECTION.
    TRY.
        TRY.
            CREATE OBJECT class1ref.
            class1ref->method_1( ).
          CATCH cx_user_exception INTO excp.
```

Exception Triggered: CX_USER_EXCEPTION

Field Names 1 - 4 Field Contents

SY-SUBRC 0   SY-TABIX 1   SY-DBCNT 0

DLT (1) (100) nadcp516 INS

Object name: {4<\PROGRAM=Z_DELTA_6_16_EXCEPT

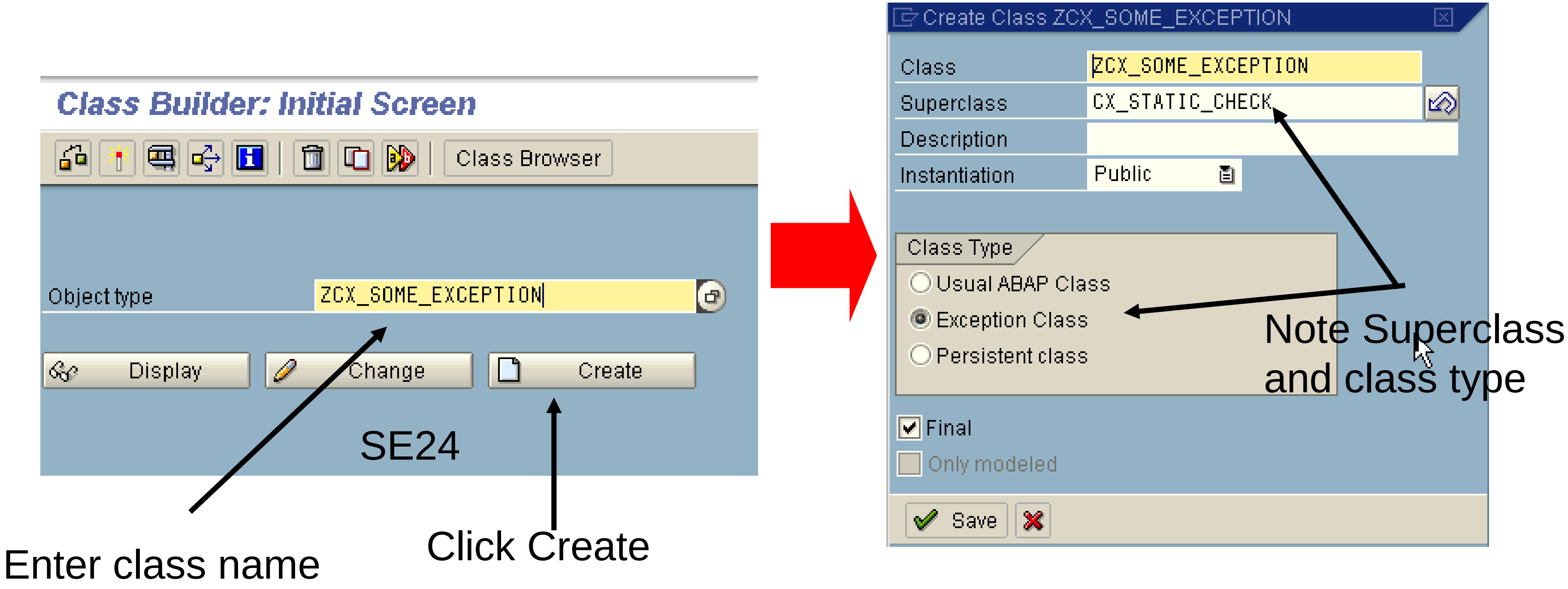
Class/Interface:

□ Static attributes   ☑ Instance attributes

☑ Public   □ Protected   □ Private

No.			Attributes/Interfaces	T	Lngth	Contents
1		●	TEXTID	C	32	16AA9A3937A9BB56E10000000A1
2		●	PREVIOUS	r	8	0<>
3		●	KERNEL_ERRID	C	30	
4		●	METH	g	8	method_2

Class-Based Exceptions – Creating a Global Exception Class (1)



Class Builder: Initial Screen

Object type ZCX_SOME_EXCEPTION

SE24

Enter class name

Click Create

Create Class ZCX_SOME_EXCEPTION

Class	ZCX_SOME_EXCEPTION
Superclass	CX_STATIC_CHECK
Description	
Instantiation	Public

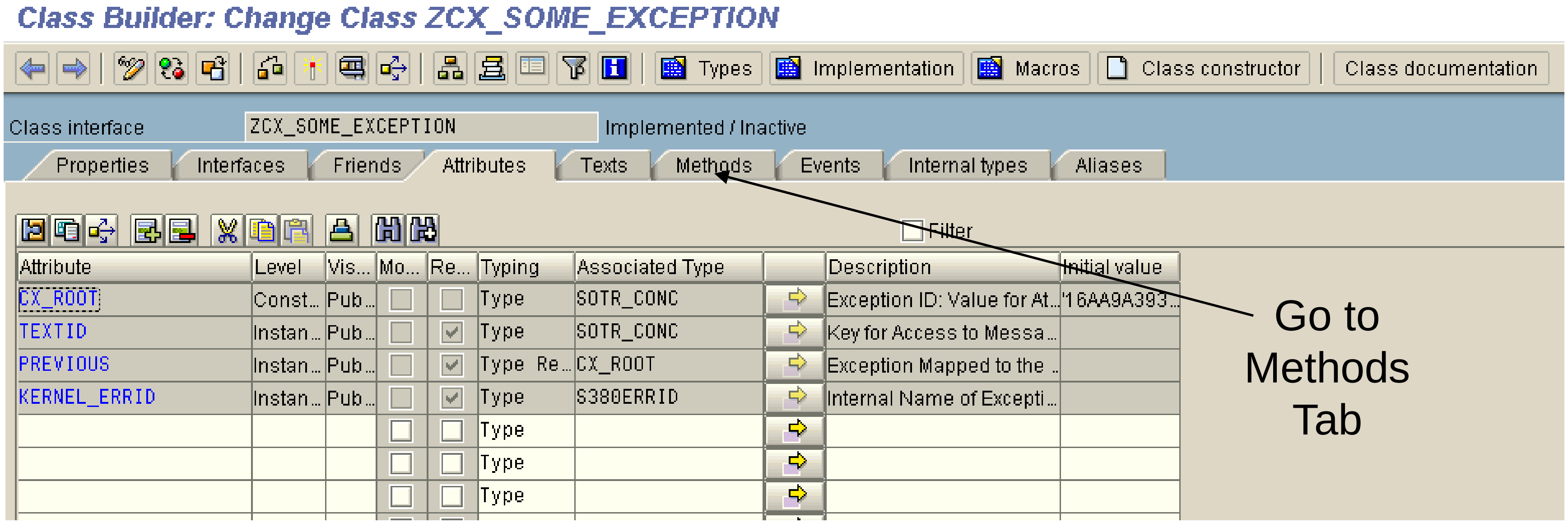
Class Type

○ Usual ABAP Class
● Exception Class
○ Persistent class

☑ Final
□ Only modeled

Note Superclass and class type

Class-Based Exceptions – Creating a Global Exception Class (2)



Class Builder: Change Class ZCX_SOME_EXCEPTION

Class interface ZCX_SOME_EXCEPTION Implemented / Inactive

Attribute	Level	Vis...	Mo...	Re...	Typing	Associated Type		Description	Initial value
CX_ROOT	Const...	Pub...	□	□	Type	SOTR_CONC	⇨	Exception ID: Value for At...	'16AA9A393...
TEXTID	Instan...	Pub...	□	☑	Type	SOTR_CONC	⇨	Key for Access to Messa...	
PREVIOUS	Instan...	Pub...	□	☑	Type Re...	CX_ROOT	⇨	Exception Mapped to the ...	
KERNEL_ERRID	Instan...	Pub...	□	☑	Type	S380ERRID	⇨	Internal Name of Excepti...	
			□	□	Type		⇨		
			□	□	Type		⇨		
			□	□	Type		⇨		

Go to Methods Tab

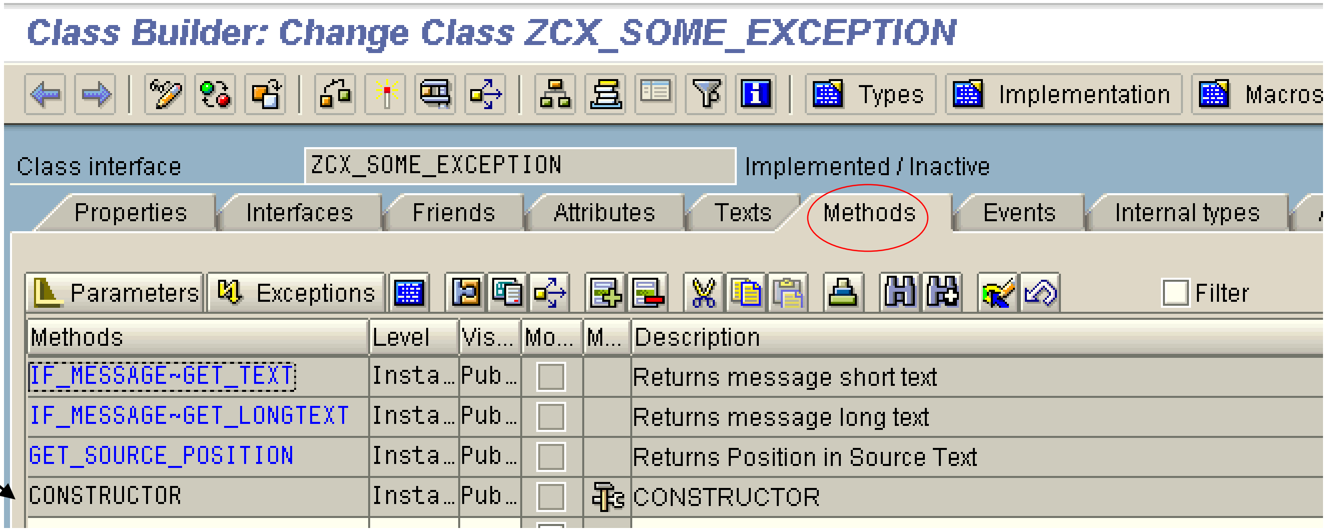
Note the 2 attributes inherited from cx_root superclass

textid – Used to define different texts for exceptions of a particular class. Affects the result of method get_text

previous – If one exception is mapped to another, this attribute can store the original exception. If a runtime error occurs, the short dump contains the texts belonging to all the exceptions in the chain

Class-Based Exceptions – Creating a Global Exception Class (3)

Double click on the constructor method to view code



Class Builder: Change Class ZCX_SOME_EXCEPTION

Class interface: ZCX_SOME_EXCEPTION Implemented / Inactive

Methods	Level	Vis...	Mo...	M...	Description
IF_MESSAGE~GET_TEXT	Insta...	Pub...	□		Returns message short text
IF_MESSAGE~GET_LONGTEXT	Insta...	Pub...	□		Returns message long text
GET_SOURCE_POSITION	Insta...	Pub...	□		Returns Position in Source Text
CONSTRUCTOR	Insta...	Pub...	□		CONSTRUCTOR

Three methods are inherited from CX_ROOT

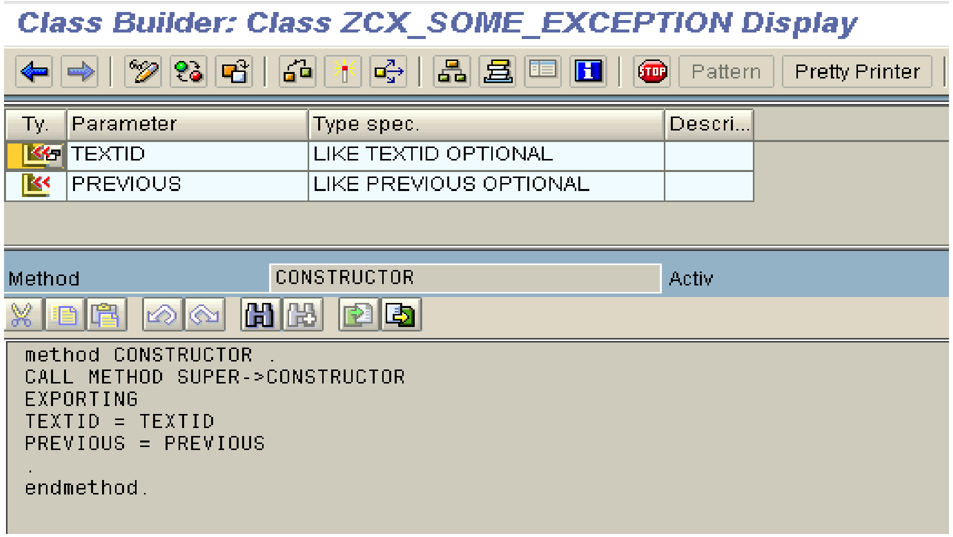
get_text, get_longtext – Returns the textual representation as a string, according to the system language of the exception

get_source_position – Returns the program name, include name, and line number reached where the exception was raised

A constructor method is automatically generated

Class-Based Exceptions – Creating a Global Exception Class (4)

Click on previous object button to return to methods tab



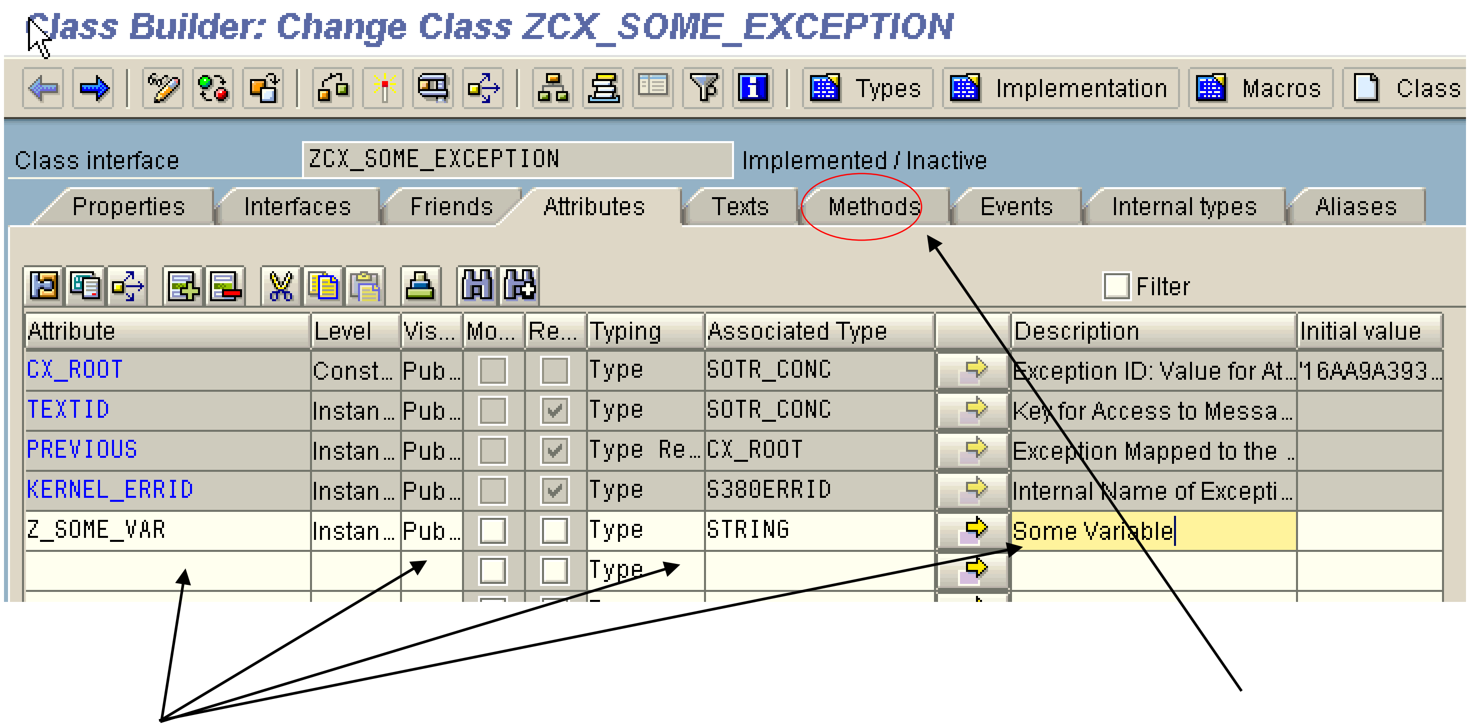
Ty.	Parameter	Type spec.	Descri...
	TEXTID	LIKE TEXTID OPTIONAL	
	PREVIOUS	LIKE PREVIOUS OPTIONAL	

Method CONSTRUCTOR Activ

```
method CONSTRUCTOR .
CALL METHOD SUPER->CONSTRUCTOR
EXPORTING
TEXTID = TEXTID
PREVIOUS = PREVIOUS
.
endmethod.
```

Call to the constructor of superclasses is automatically generated

Class-Based Exceptions – Creating a Global Exception Class (5)



Class Builder: Change Class ZCX_SOME_EXCEPTION

Class interface ZCX_SOME_EXCEPTION Implemented / Inactive

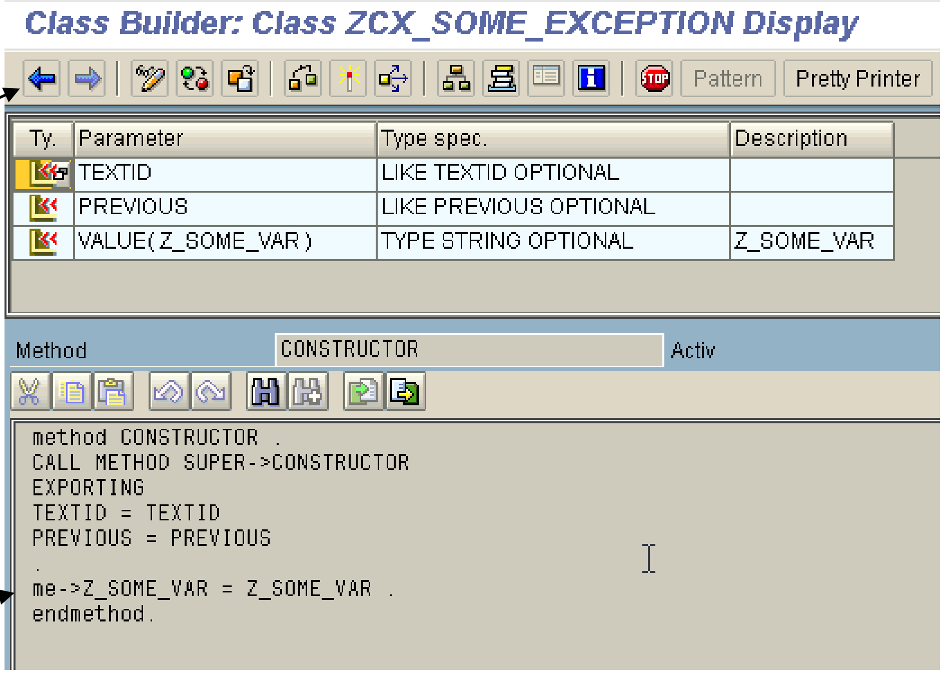
Attribute	Level	Vis...	Mo...	Re...	Typing	Associated Type		Description	Initial value
CX_ROOT	Const...	Pub...	□	□	Type	SOTR_CONC	⇨	Exception ID: Value for At...	'16AA9A393...
TEXTID	Instan...	Pub...	□	☑	Type	SOTR_CONC	⇨	Key for Access to Messa...	
PREVIOUS	Instan...	Pub...	□	☑	Type Re...	CX_ROOT	⇨	Exception Mapped to the ...	
KERNEL_ERRID	Instan...	Pub...	□	☑	Type	S380ERRID	⇨	Internal Name of Excepti...	
Z_SOME_VAR	Instan...	Pub...	□	□	Type	STRING	⇨	Some Variable	
			□	□	Type		⇨		

- First add an attribute to the error class and activate the class

- Then return to the methods tab and click on the constructor again

Class-Based Exceptions – Creating a Global Exception Class (6)

Click on previous object button to return to methods tab



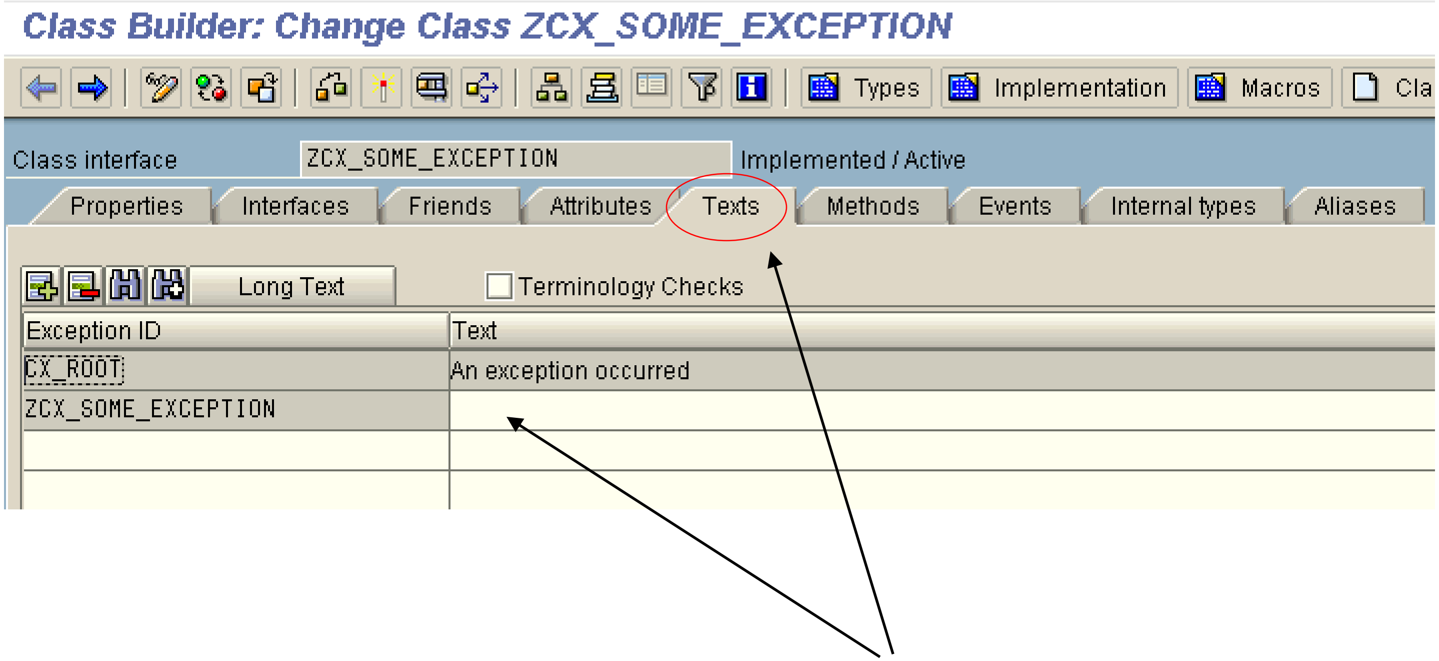
Ty.	Parameter	Type spec.	Description
	TEXTID	LIKE TEXTID OPTIONAL	
	PREVIOUS	LIKE PREVIOUS OPTIONAL	
	VALUE(Z_SOME_VAR)	TYPE STRING OPTIONAL	Z_SOME_VAR

Method: CONSTRUCTOR

```
method CONSTRUCTOR .
CALL METHOD SUPER->CONSTRUCTOR
EXPORTING
TEXTID = TEXTID
PREVIOUS = PREVIOUS
.
me->Z_SOME_VAR = Z_SOME_VAR .
endmethod.
```

A line has been added to the constructor to initialize the new attribute. This attribute will be available in the error object at runtime and will contain the value that is passed to the constructor when the exception is raised

Class-Based Exceptions – Creating a Global Exception Class (7)


Class Builder: Change Class ZCX_SOME_EXCEPTION

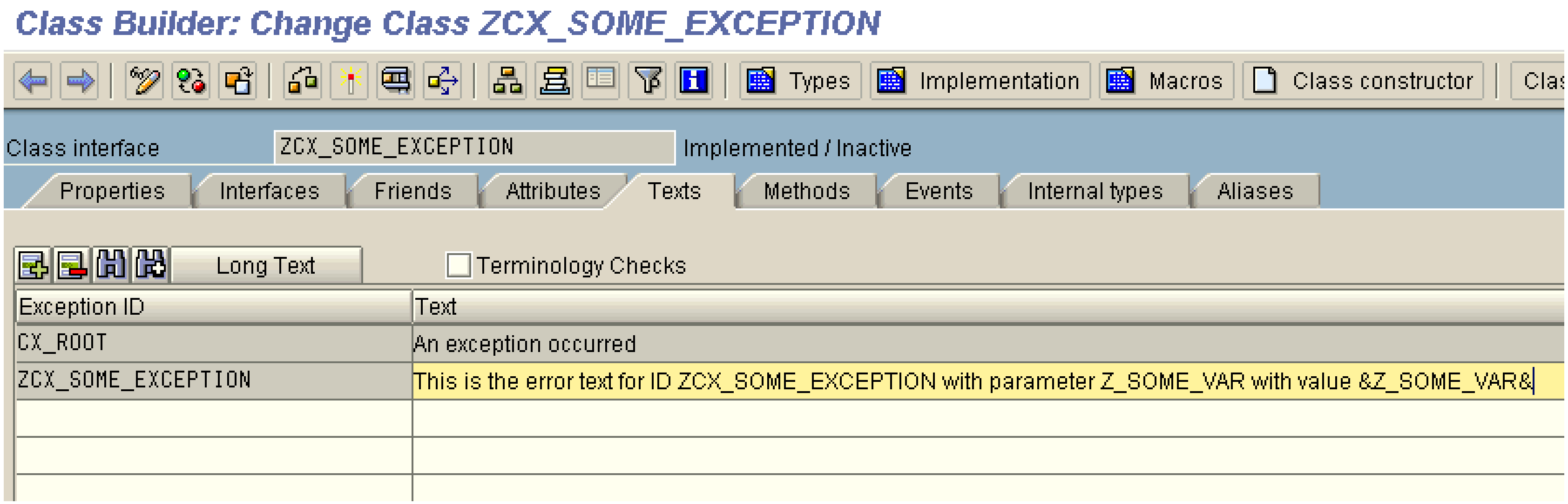
Class interface ZCX_SOME_EXCEPTION Implemented / Active

□ Terminology Checks

Exception ID	Text
CX_ROOT	An exception occurred
ZCX_SOME_EXCEPTION	

Go to the Texts tab and add a text for the exception ID.

Class-Based Exceptions – Creating a Global Exception Class (8)



Class Builder: Change Class ZCX_SOME_EXCEPTION

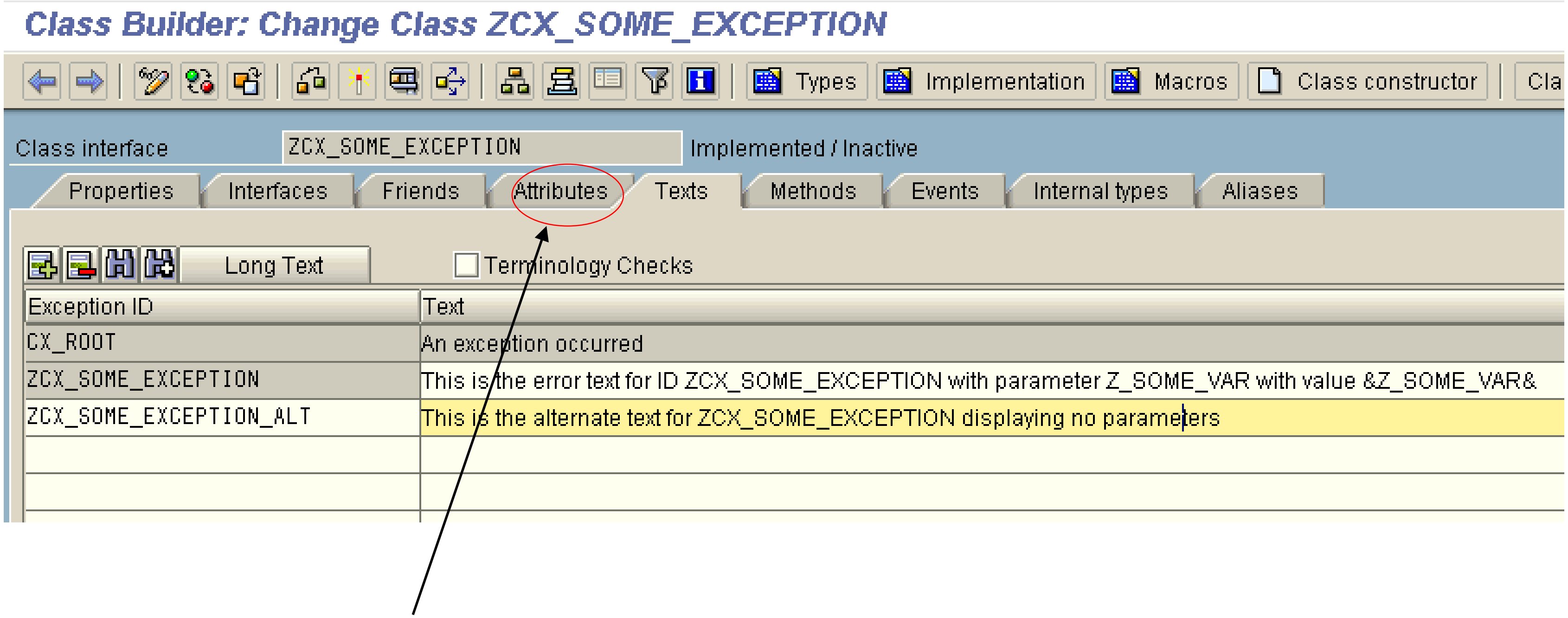
Class interface ZCX_SOME_EXCEPTION Implemented / Inactive

□ Terminology Checks

Exception ID	Text
CX_ROOT	An exception occurred
ZCX_SOME_EXCEPTION	This is the error text for ID ZCX_SOME_EXCEPTION with parameter Z_SOME_VAR with value &Z_SOME_VAR&

- The texts are stored in the Online Text Repository (OTR). The exception object contains only a key that identifies the text (with system language)
- The default text has the same name as the name of the exception class, in this case ZCX_SOME_EXCEPTION.
- You might wish to create an alternate text for the exception. That text can be entered on this screen with a new exception ID and can be displayed by passing this value to the parameter textid of the exception constructor.

Class-Based Exceptions – Creating a Global Exception Class (9)



Class Builder: Change Class ZCX_SOME_EXCEPTION

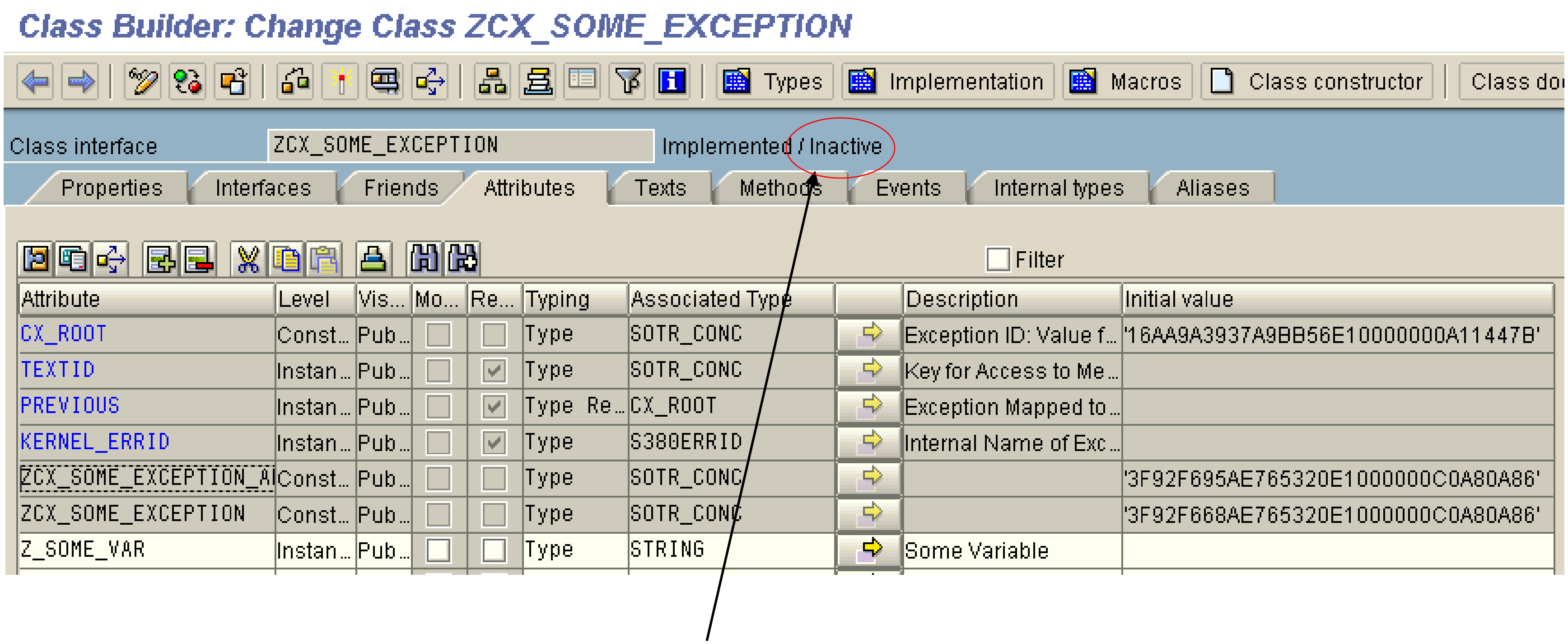
Class interface ZCX_SOME_EXCEPTION Implemented / Inactive

□ Terminology Checks

Exception ID	Text
CX_ROOT	An exception occurred
ZCX_SOME_EXCEPTION	This is the error text for ID ZCX_SOME_EXCEPTION with parameter Z_SOME_VAR with value &Z_SOME_VAR&
ZCX_SOME_EXCEPTION_ALT	This is the alternate text for ZCX_SOME_EXCEPTION displaying no parameters

After performing a syntax check and adding the texts to the OTR, return to the Attributes tab

Class-Based Exceptions – Creating a Global Exception Class (10)



Class Builder: Change Class ZCX_SOME_EXCEPTION

Class interface ZCX_SOME_EXCEPTION Implemented / Inactive

Attribute	Level	Vis...	Mo...	Re...	Typing	Associated Type		Description	Initial value
CX_ROOT	Const...	Pub...	□	□	Type	SOTR_CONC	⇨	Exception ID: Value f...	'16AA9A3937A9BB56E10000000A11447B'
TEXTID	Instan...	Pub...	□	☑	Type	SOTR_CONC	⇨	Key for Access to Me...	
PREVIOUS	Instan...	Pub...	□	☑	Type Re...	CX_ROOT	⇨	Exception Mapped to...	
KERNEL_ERRID	Instan...	Pub...	□	☑	Type	S380ERRID	⇨	Internal Name of Exc...	
ZCX_SOME_EXCEPTION_A	Const...	Pub...	□	□	Type	SOTR_CONC	⇨		'3F92F695AE765320E1000000C0A80A86'
ZCX_SOME_EXCEPTION	Const...	Pub...	□	□	Type	SOTR_CONC	⇨		'3F92F668AE765320E1000000C0A80A86'
Z_SOME_VAR	Instan...	Pub...	□	□	Type	STRING	⇨	Some Variable	

Don’t forget to activate the object!

Note that the text ID’s have been added to the attributes page as class constants