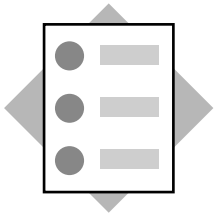

ABAP Objects
- Introduction
A Workshop from the
ABAP Language Group

Workshop Goals

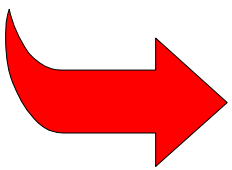


- Position of ABAP Objects within the R/3 System
- Overview of the syntax of ABAP Objects
- Working with existing classes and interfaces
- Defining classes and interfaces
- Creating objects
- Reacting to events
- Understanding the polymorphism provided by interfaces and inheritance

Workshop Goals

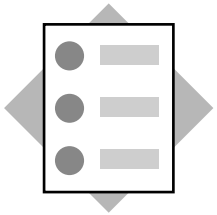
This is not a comprehensive course in object-oriented programming

- There is more to object-oriented development than just object-oriented programming. It has logical advantages that are independent of the concrete implementation.
- The most important (and most time-consuming) part of a real object-oriented application is the object-oriented modeling.



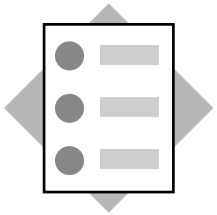
The OO Rollout Group provides another ABAP Objects course

Contents



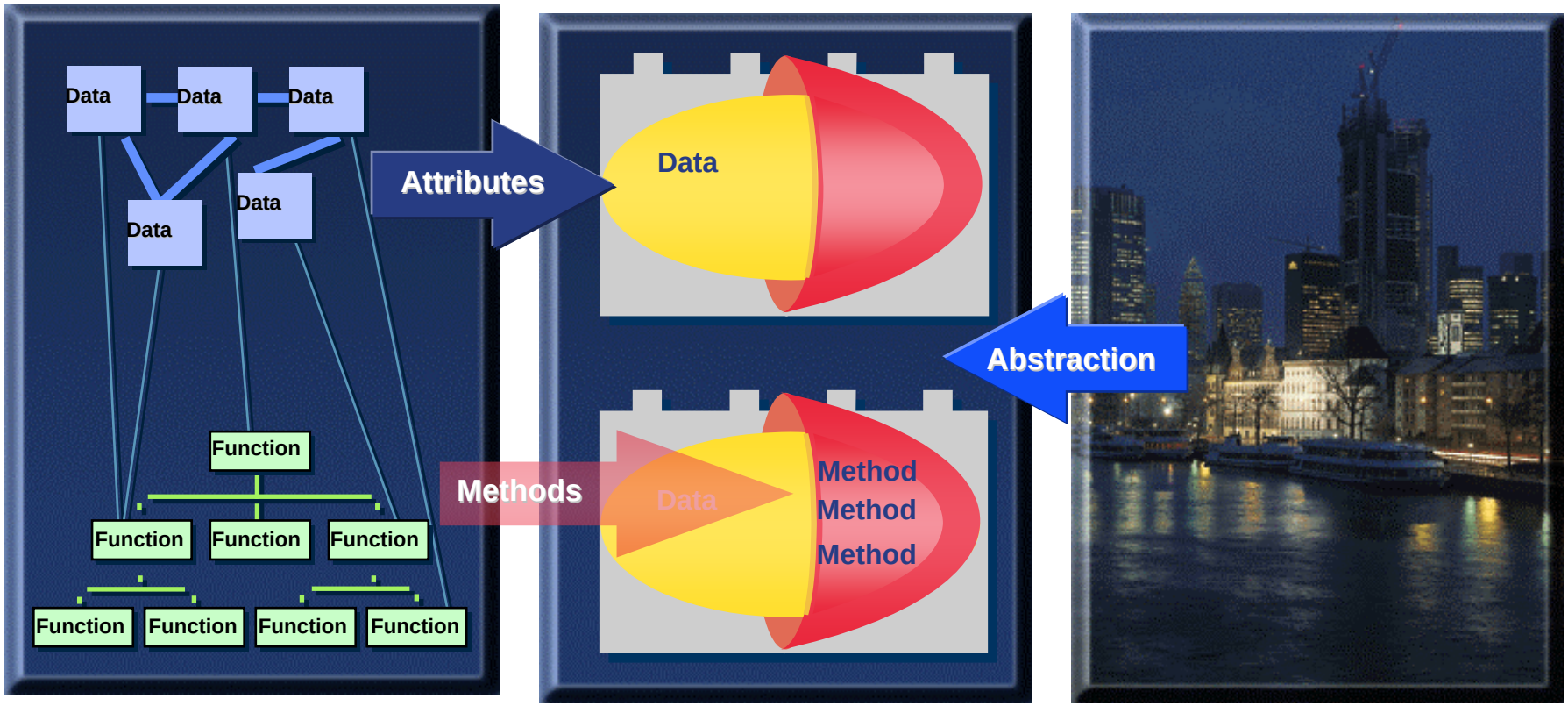
- Introduction
- From Function Groups to Classes
- Classes
- Objects
- Events
- Interfaces
- Inheritance
- Using Global Classes
- Exercises

Introduction



- Object orientation
- Objects
- ABAP Objects

Object Orientiation

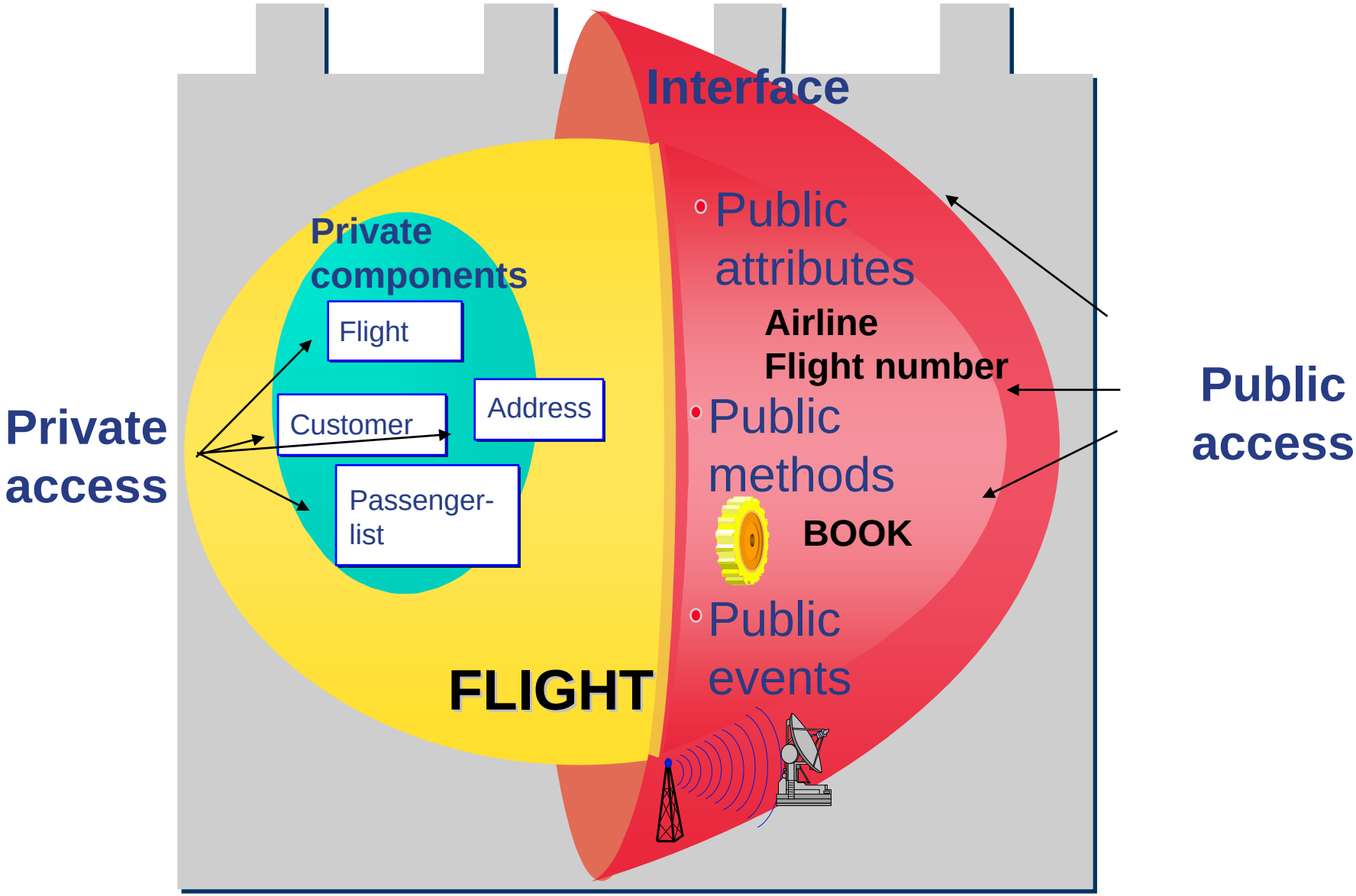


Functions and data
Data model as an abstraction of the real world

Software objects
Object model as an abstraction of the real world

Real-world objects

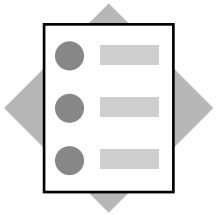
Objects


Interface
Private components
Flight
Customer
Address
Passenger-list
FLIGHT
Public attributes
Airline
Flight number
Public methods
BOOK
Public events
Private access
Public access

ABAP Objects

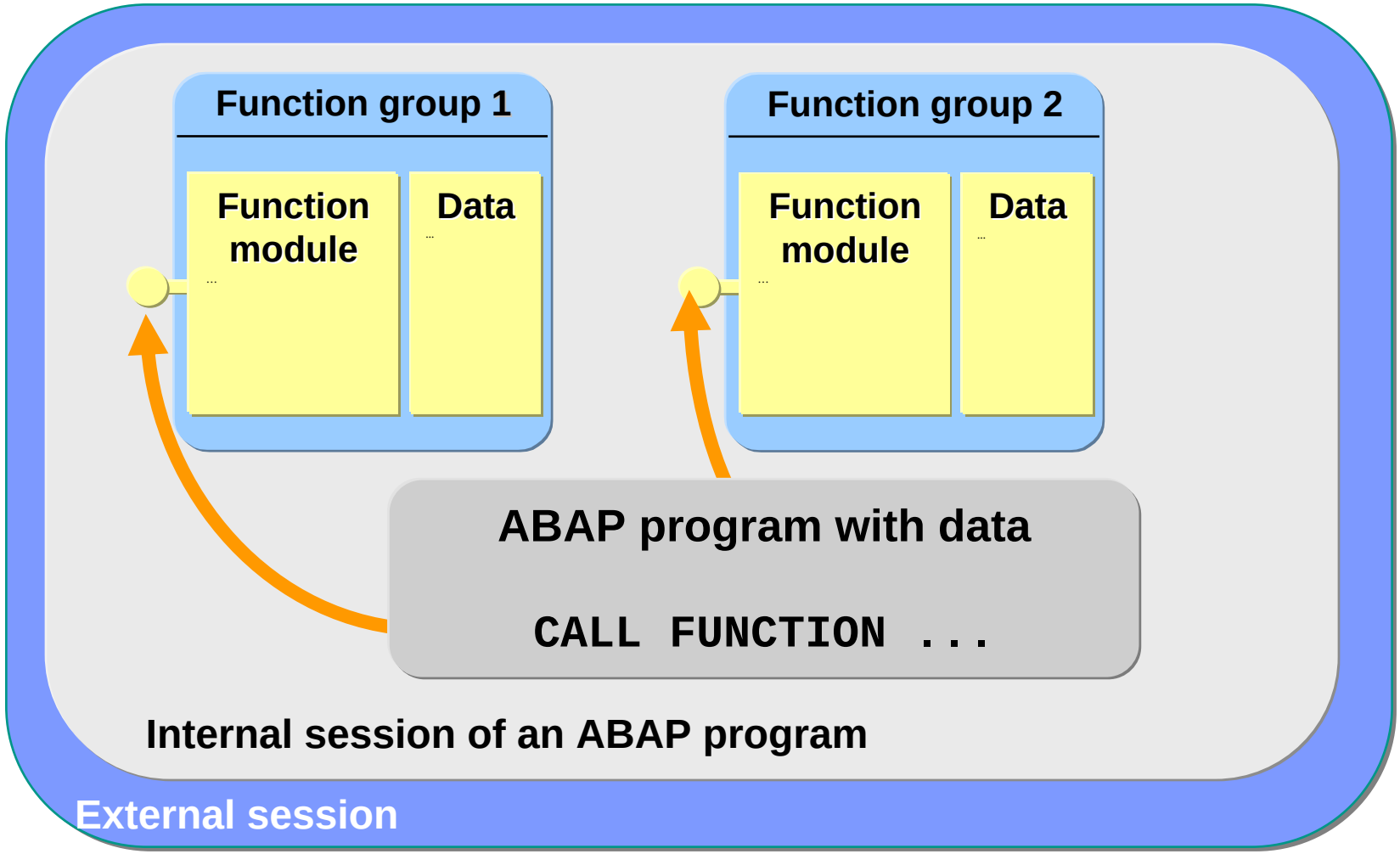
- ABAP Objects is an upwards-compatible extension of the existing ABAP language
- You can use existing ABAP statements within ABAP Objects
- You can use ABAP Objects within existing programs
- ABAP Objects is fully integrated in the ABAP Debugger

From Function Groups to Classes



- **Instances of function groups as objects**
- **Example: Function group as counter**

Instances of Function Groups as Objects

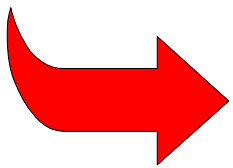


Function Group as Counter - Definition

```
FUNCTION-POOL COUNTER.

DATA COUNT TYPE I.

FUNCTION SET_COUNTER.
* Local Interface IMPORTING VALUE(SET_VALUE)
  COUNT = SET_VALUE.
ENDFUNCTION.

FUNCTION INCREMENT_COUNTER.
  COUNT = COUNT + 1.
ENDFUNCTION.

FUNCTION GET_COUNTER.
* Local Interface: EXPORTING VALUE(GET_VALUE)
  GET_VALUE = COUNT.
ENDFUNCTION.
```

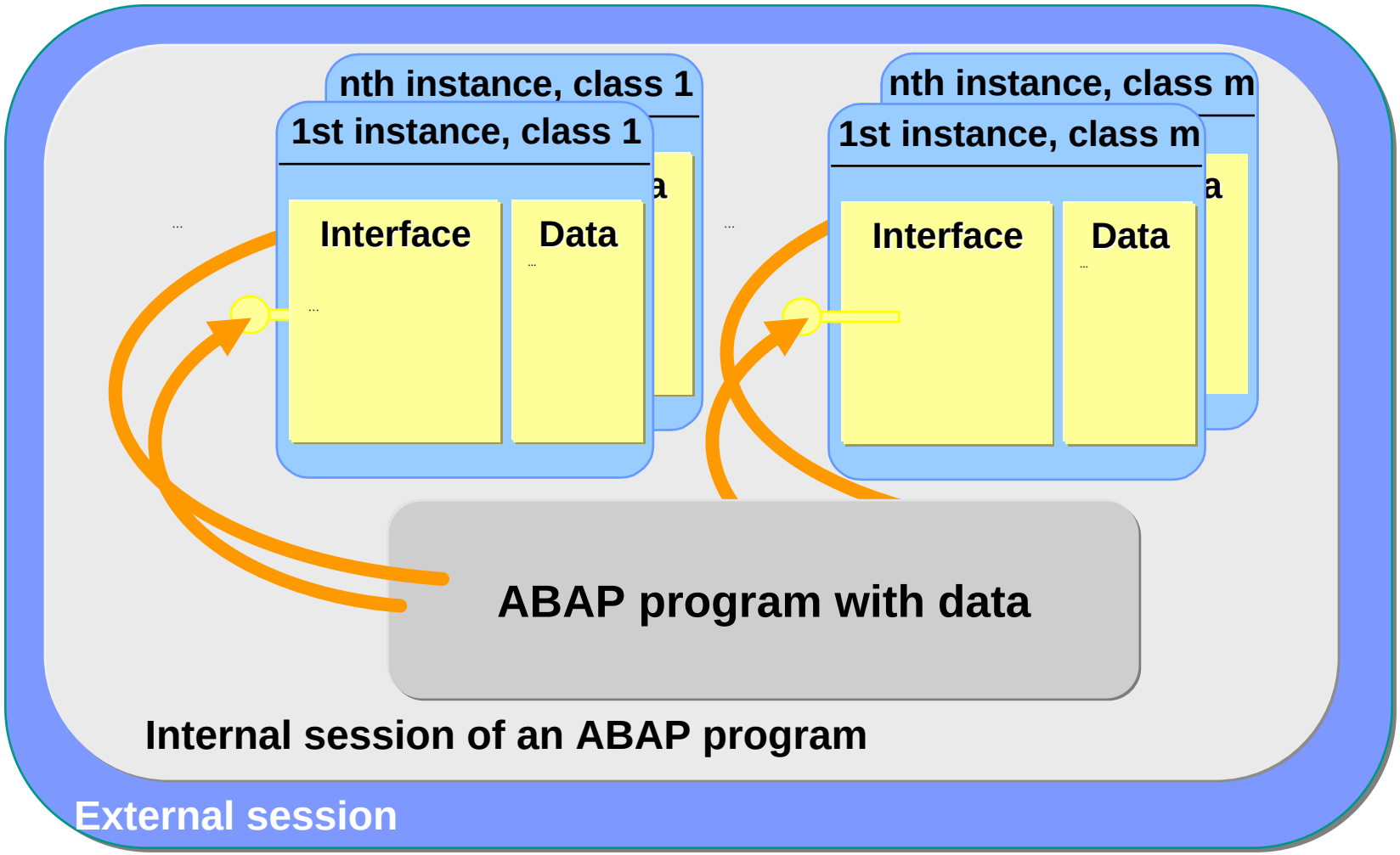
Function Group as Counter - Call

```
DATA NUMBER TYPE I VALUE 5.

CALL FUNCTION 'SET_COUNTER' EXPORTING
                            SET_VALUE = NUMBER.

DO 3 TIMES.
  CALL FUNCTION 'INCREMENT_COUNTER'.
ENDDO.

CALL FUNCTION 'GET_COUNTER' IMPORTING
                            GET_VALUE = NUMBER.
```

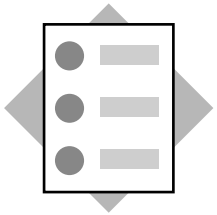


NUMBER has value 8

Classes Generalize Function Groups



Classes, References, and Objects



- **Example: Class as counter**
- **Reference variables**
- **Creating objects**
- **Calling methods**
- **Working with references**

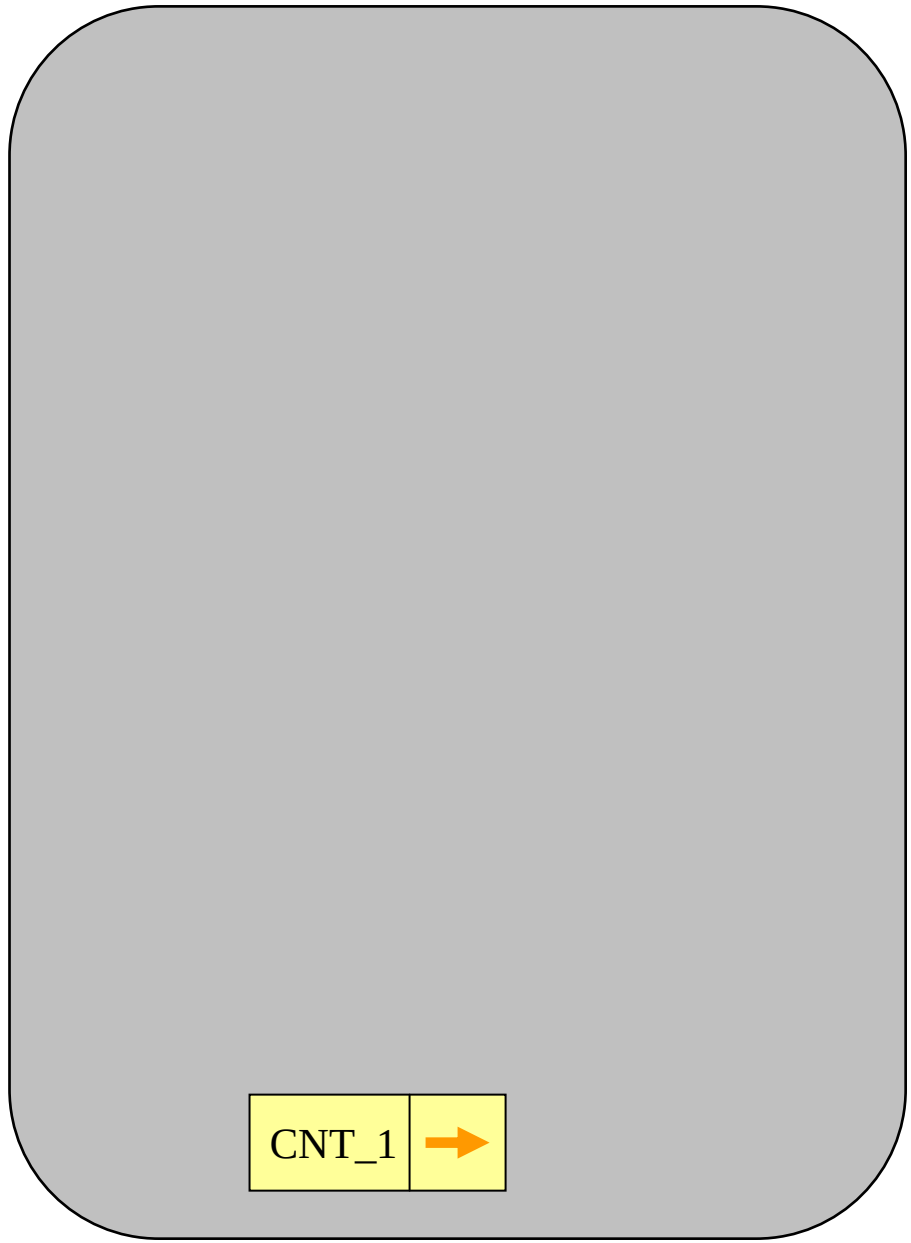
Example: Class as Counter

```
CLASS counter DEFINITION.
  PUBLIC SECTION.
    METHODS: set IMPORTING
                   VALUE(set_value) TYPE i,
             increment,
             get EXPORTING
                   VALUE(get_value) TYPE i.
  PRIVATE SECTION.
    DATA count TYPE i.
ENDCLASS.
```

```
CLASS counter IMPLEMENTATION.
  METHOD set.
    count = set_value.
  ENDMETHOD.
  METHOD increment.
    count = count + 1.
  ENDMETHOD.
  METHOD get.
    get_value = count.
  ENDMETHOD.
ENDCLASS.
```

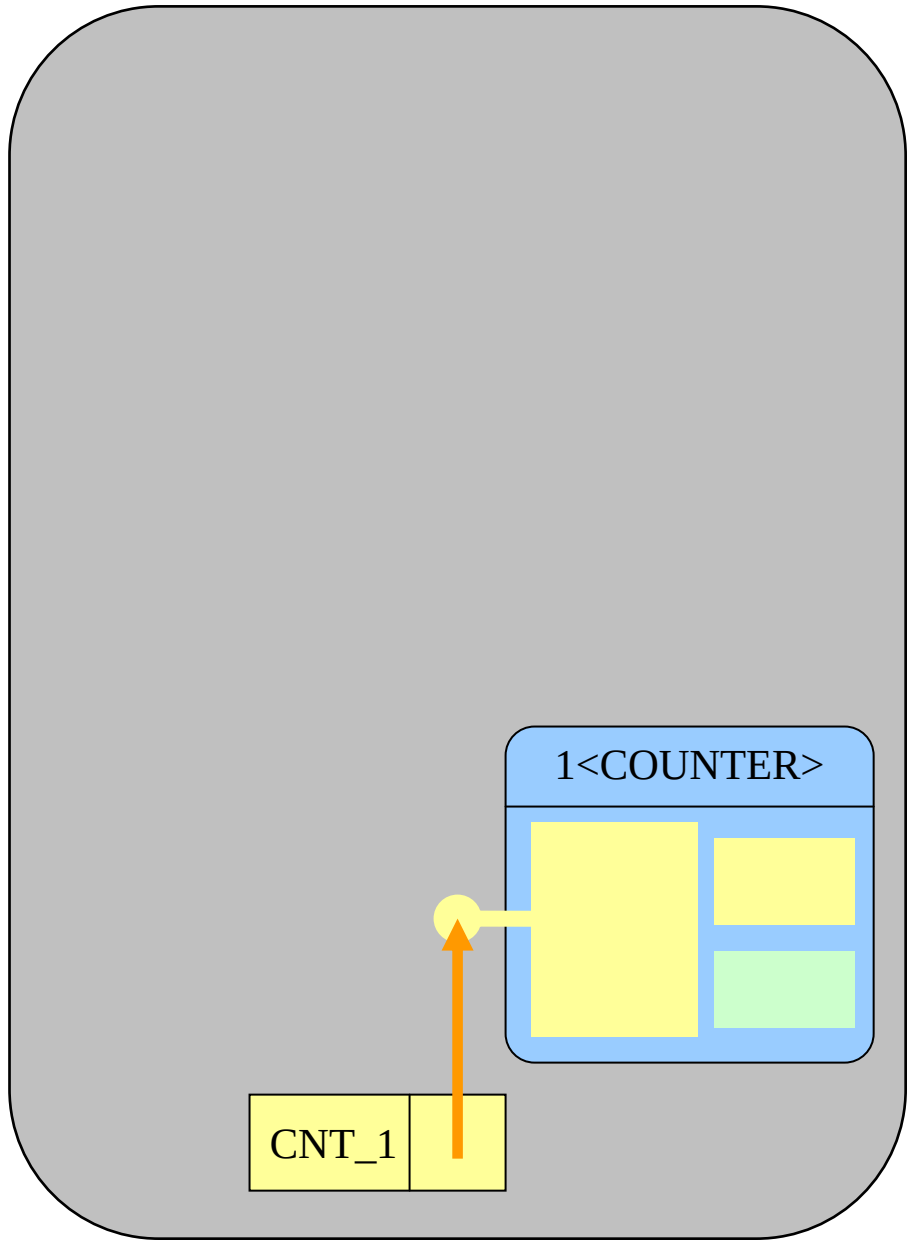
Reference Variables

```
DATA: cnt_1 TYPE REF TO counter.
```



Creating an Object

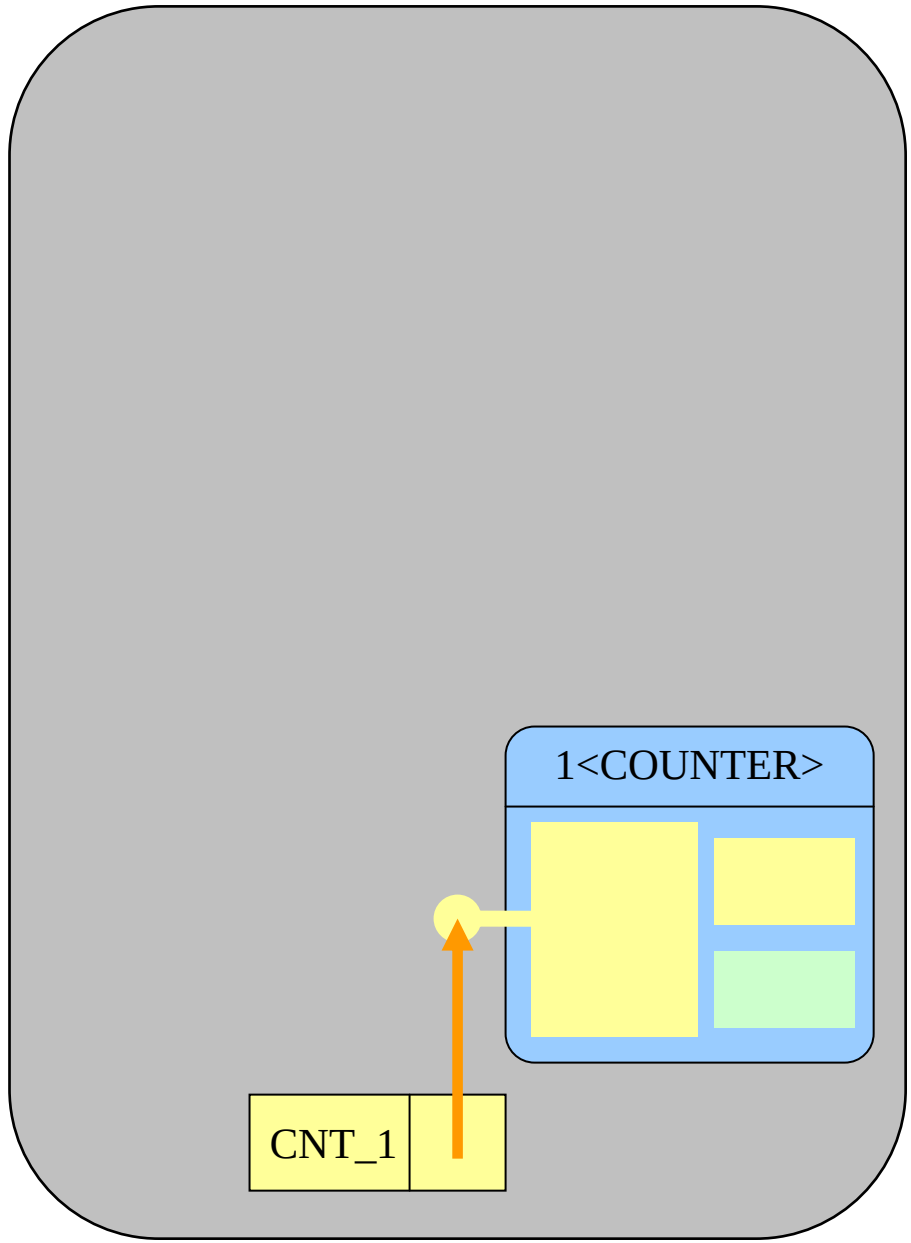
```
DATA: cnt_1 TYPE REF TO counter.


CREATE OBJECT cnt_1 TYPE counter.
```



Calling Methods

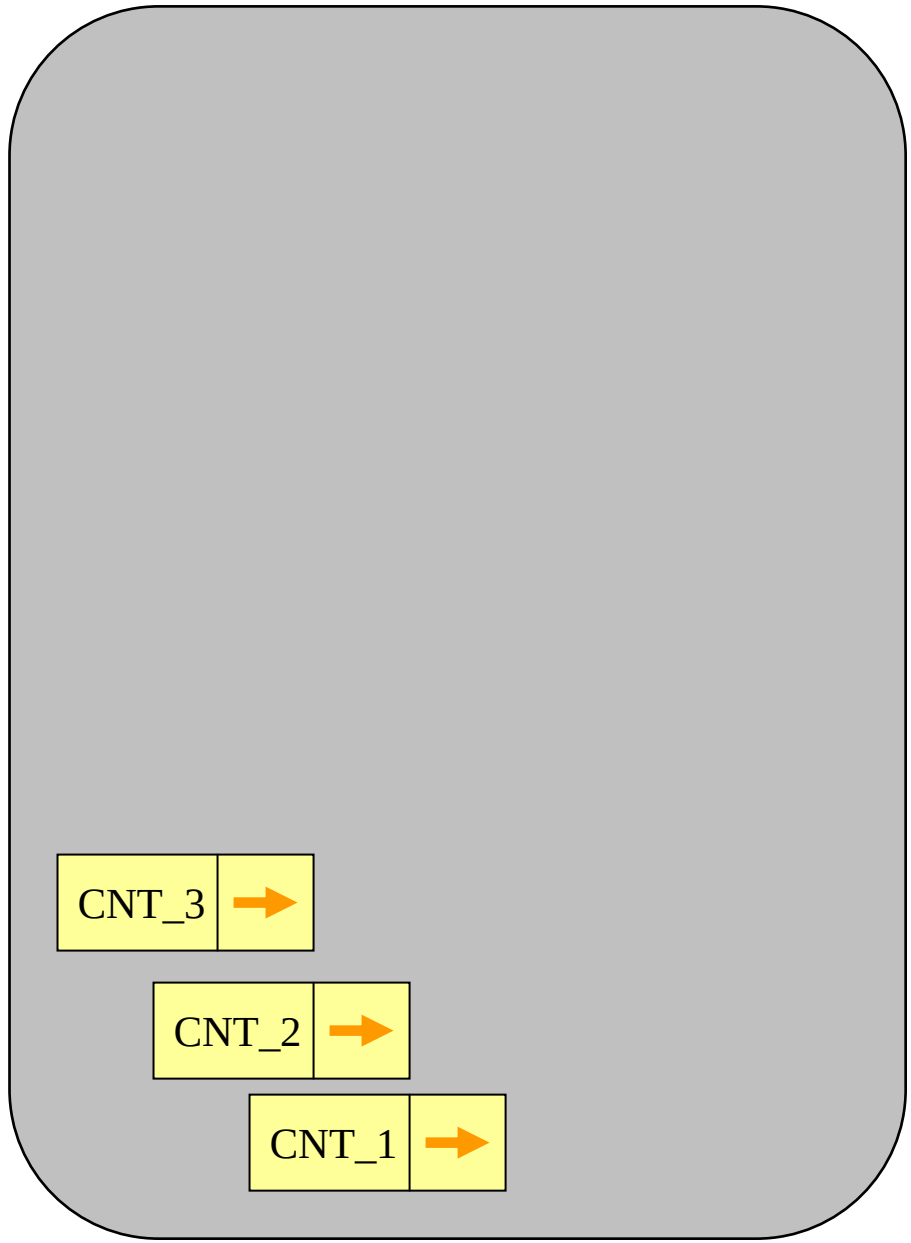
```
DATA: cnt_1 TYPE REF TO counter.

DATA number TYPE I VALUE 5.

CREATE OBJECT cnt_1 TYPE counter.

CALL METHOD cnt_1->set
     EXPORTING set_value = number.

DO 3 TIMES.
  CALL METHOD
       cnt_1->increment.
ENDDO.

CALL METHOD cnt_1->get
     IMPORTING get_value = number.
```

NUMBER has the value 8



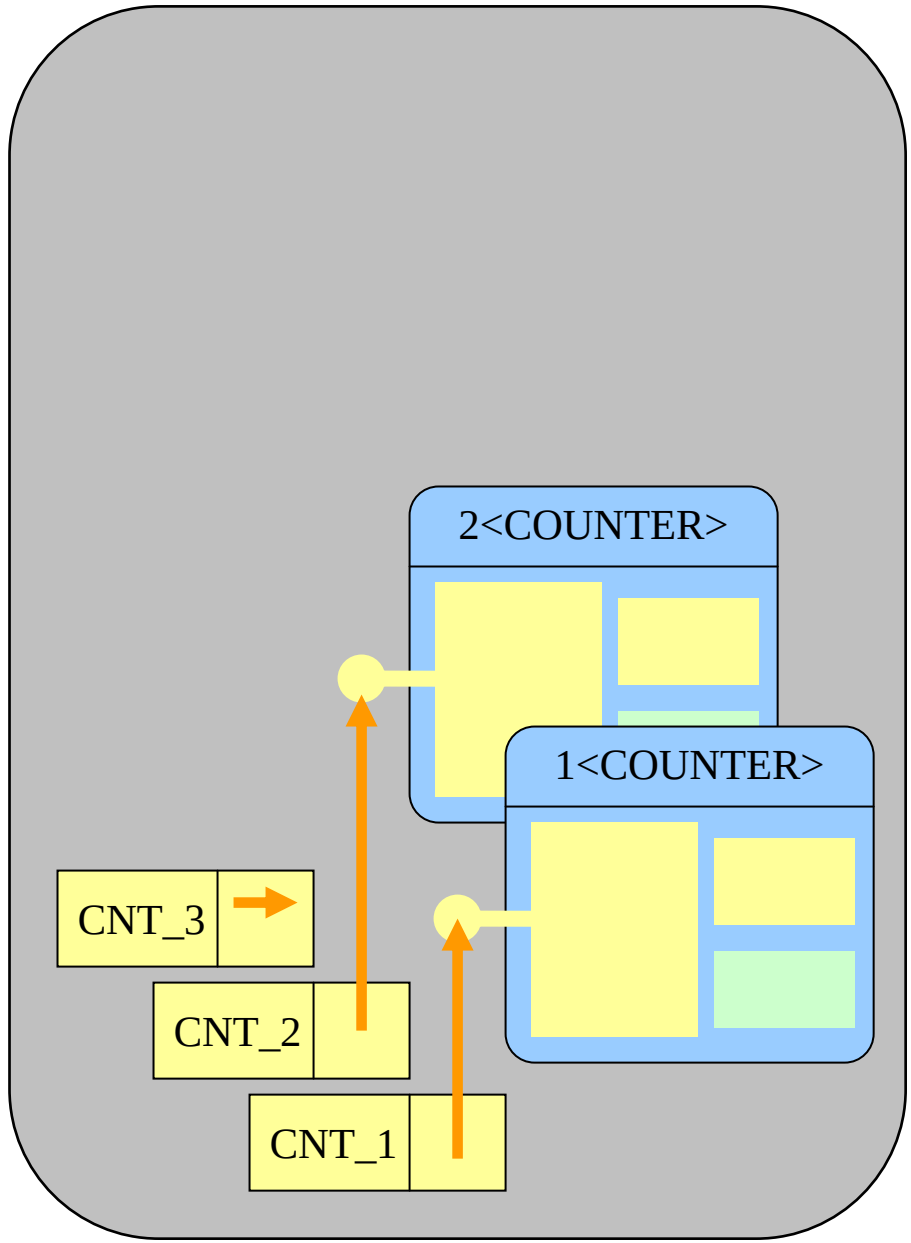
Several Reference Variables

```
DATA: cnt_1 TYPE REF TO counter,
      cnt_2 TYPE REF TO counter,
      cnt_3 TYPE REF TO counter.
```



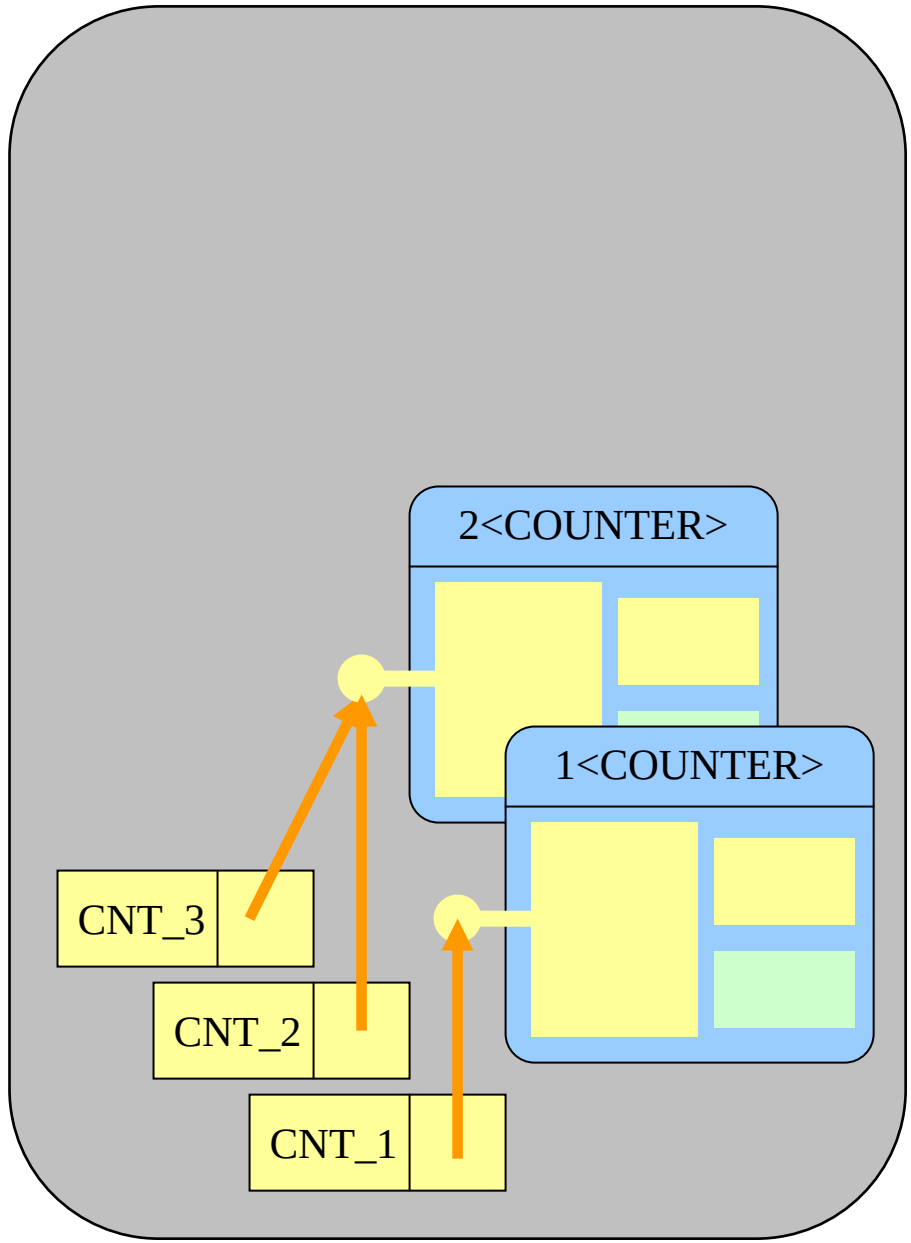
Several Objects

```
DATA: cnt_1 TYPE REF TO counter,
      cnt_2 TYPE REF TO counter,
      cnt_3 TYPE REF TO counter.

CREATE OBJECT: cnt_1,
               cnt_2.
```



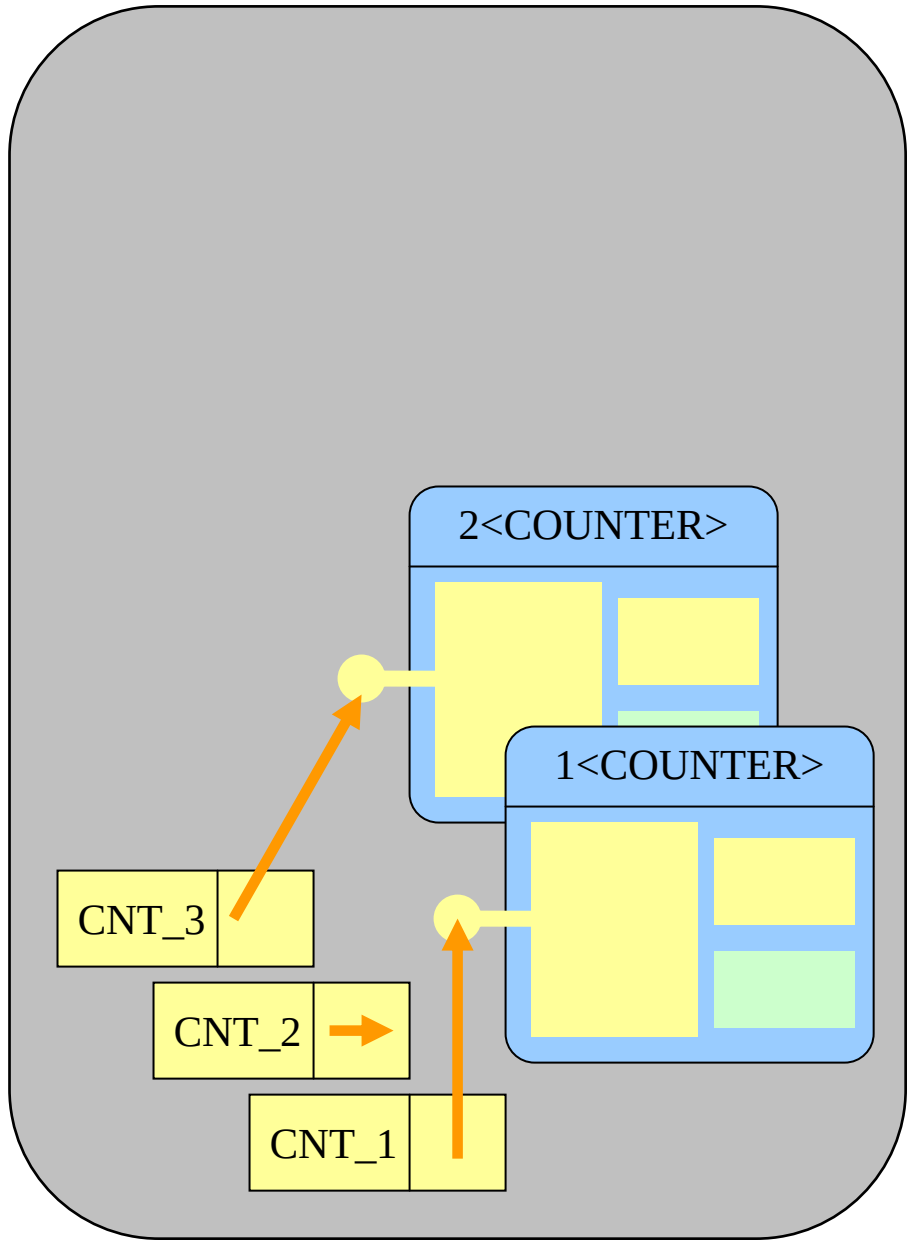
Assigning Reference Variables

```
DATA: cnt_1 TYPE REF TO counter,
      cnt_2 TYPE REF TO counter,
      cnt_3 TYPE REF TO counter.

CREATE OBJECT: cnt_1,
               cnt_2.

MOVE cnt_2 TO cnt_3.
```



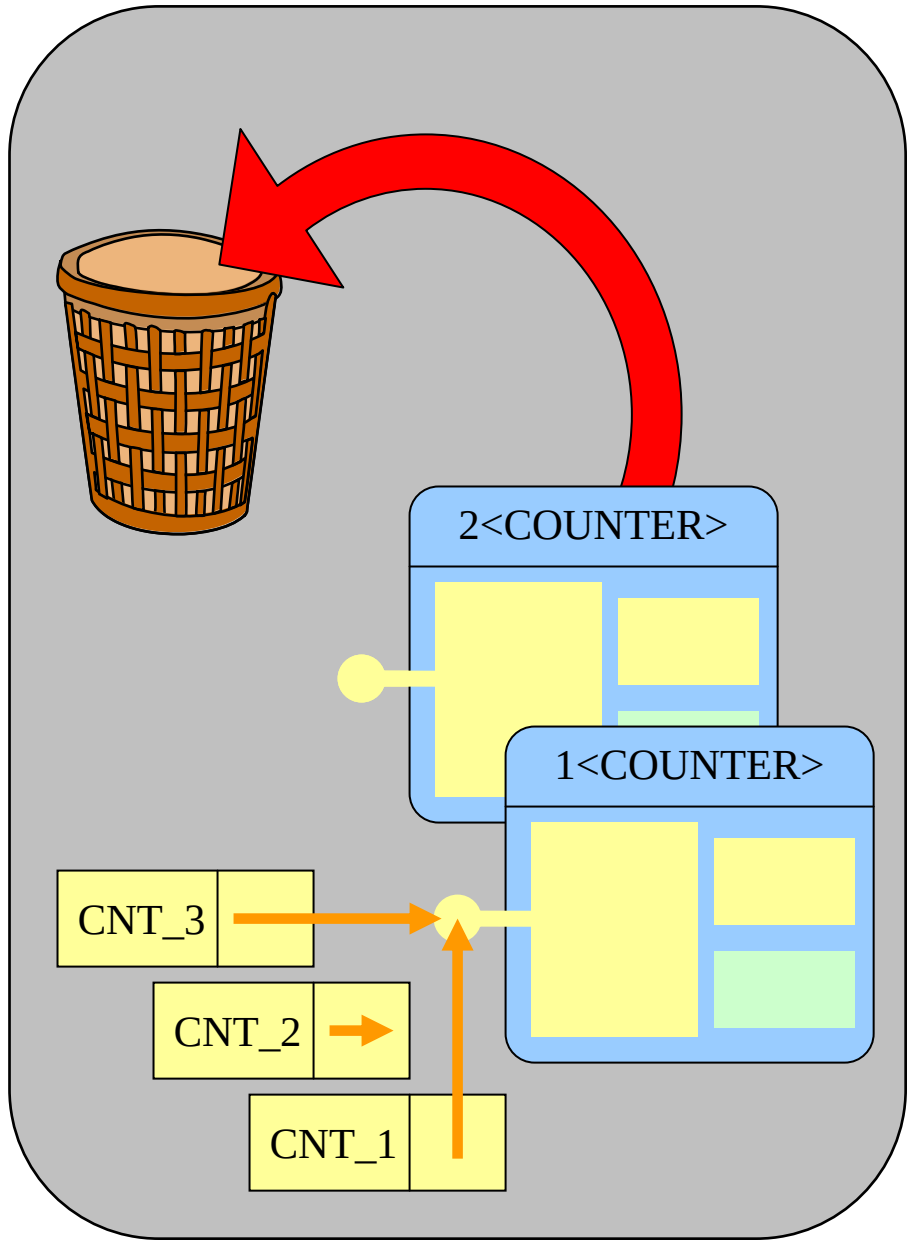
Deleting Reference Variables

```
DATA: cnt_1 TYPE REF TO counter,
      cnt_2 TYPE REF TO counter,
      cnt_3 TYPE REF TO counter.

CREATE OBJECT: cnt_1,
               cnt_2.

MOVE cnt_2 TO cnt_3.

CLEAR cnt_2.
```



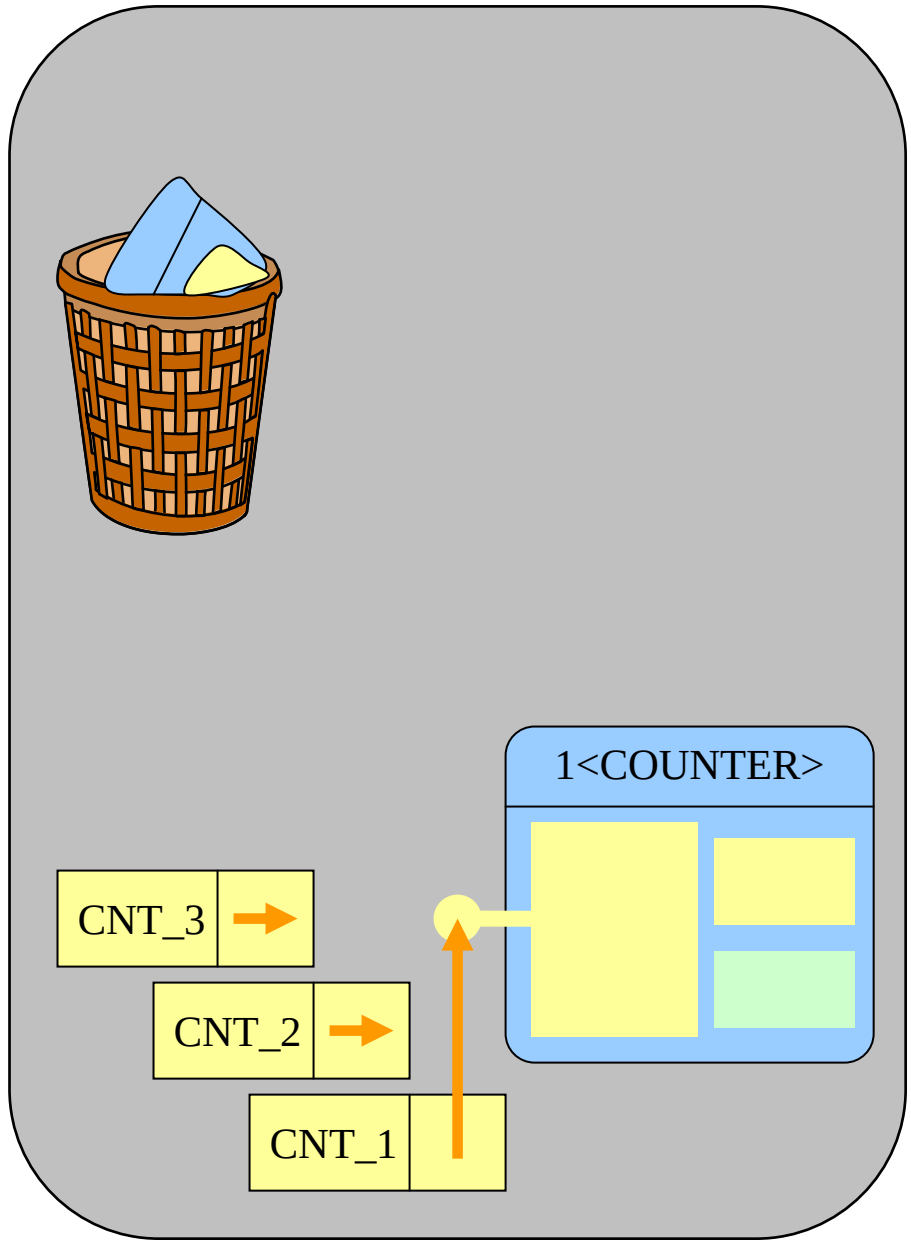
Garbage Collection

```
DATA: cnt_1 TYPE REF TO counter,
      cnt_2 TYPE REF TO counter,
      cnt_3 TYPE REF TO counter.

CREATE OBJECT: cnt_1,
               cnt_2.

MOVE cnt_2 TO cnt_3.

CLEAR cnt_2.

cnt_3 = cnt_1.
```



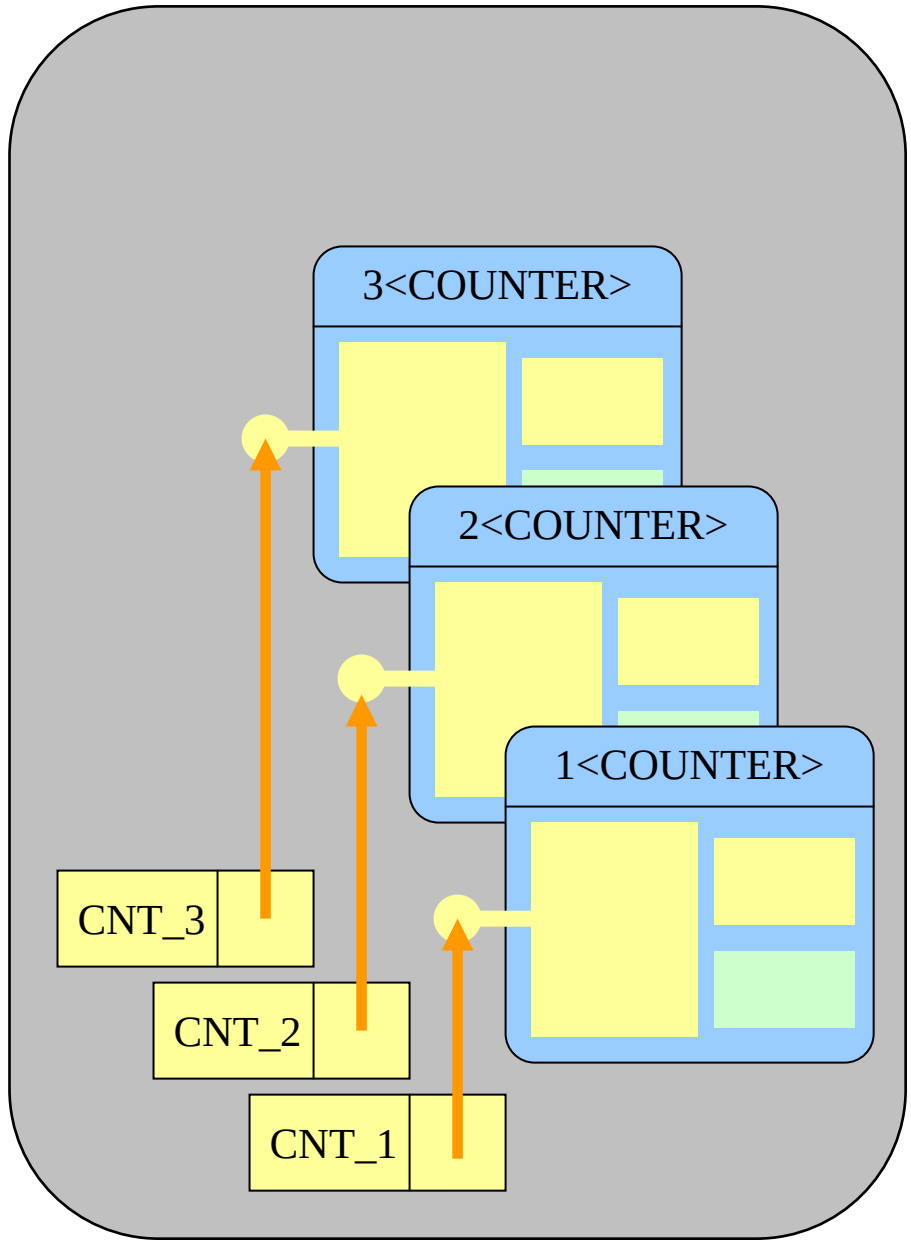
Garbage Collection

```
DATA: cnt_1 TYPE REF TO counter,
      cnt_2 TYPE REF TO counter,
      cnt_3 TYPE REF TO counter.

CREATE OBJECT: cnt_1,
               cnt_2.

MOVE cnt_2 TO cnt_3.

CLEAR cnt_2.

cnt_3 = cnt_1.

CLEAR cnt_3.
```



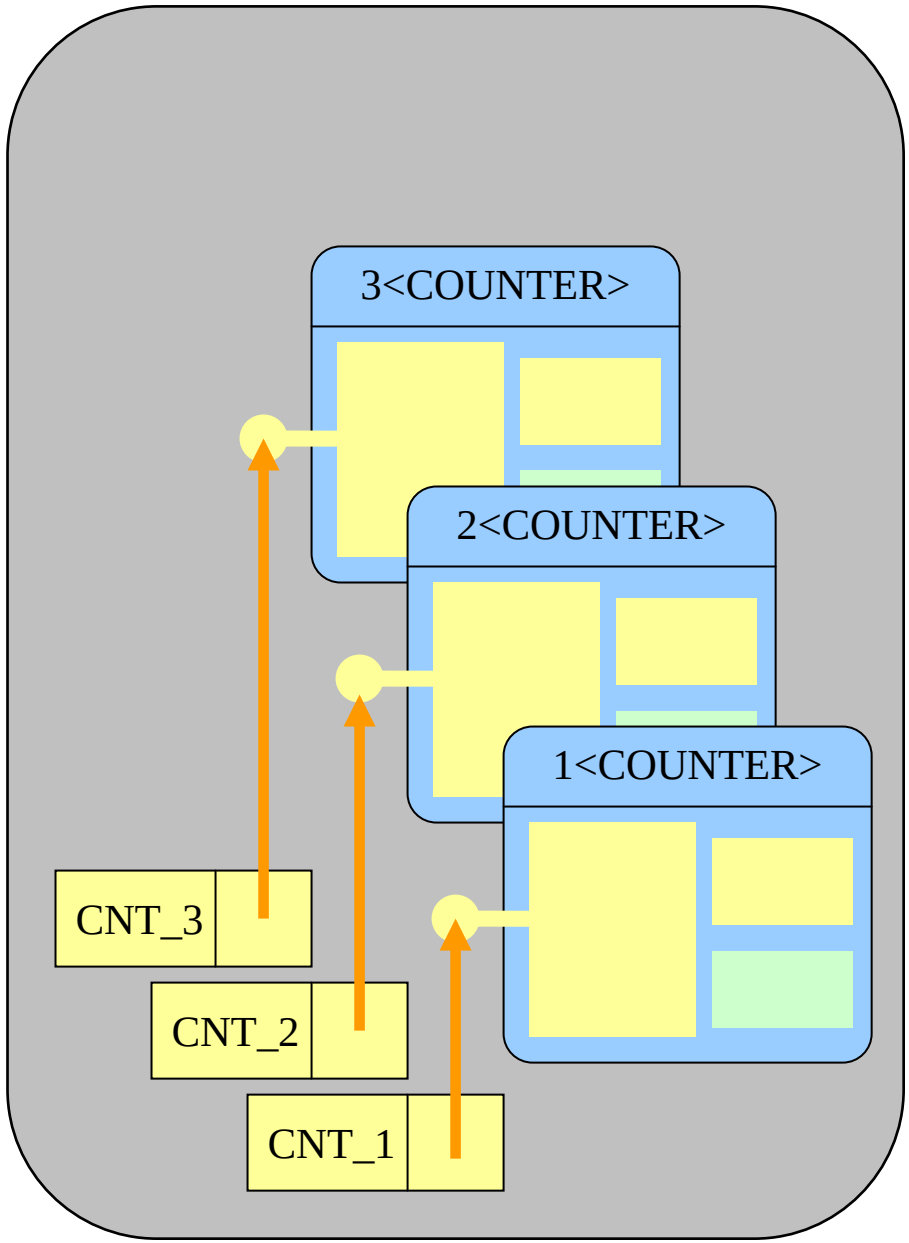
New Objects

```
DATA: cnt_1 TYPE REF TO counter,
      cnt_2 TYPE REF TO counter,
      cnt_3 TYPE REF TO counter.

CREATE OBJECT: cnt_1,
               cnt_2.

MOVE cnt_2 TO cnt_3.

CLEAR cnt_2.

cnt_3 = cnt_1.

CLEAR cnt_3.

CREATE OBJECT: cnt_2,
               cnt_3.
```



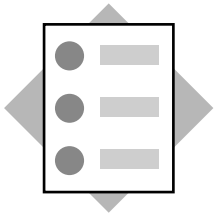
Methods of Several Objects

```
DATA: cnt_1 TYPE REF TO counter,
      cnt_2 TYPE REF TO counter,
      cnt_3 TYPE REF TO counter.

CREATE OBJECT: cnt_1,
               cnt_2,
               cnt_3.

CALL METHOD cnt_1->set
     EXPORTING set_value = 1.

CALL METHOD cnt_2->set
     EXPORTING set_value = 10.

CALL METHOD cnt_3->set
     EXPORTING set_value = 100.
```



The value of COUNT is different in each object

Objects: Summary



- Declaring reference variables

```
DATA: ref1 TYPE REF TO class,
      ref2 TYPE REF TO class.
```

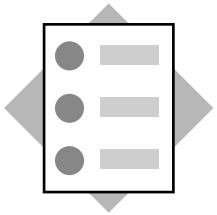
- Creating objects

```
CREATE OBJECT: ref1, ref2.
```

- Accessing attributes and methods

```
x = ref1->attr + ref2->attr.
CALL METHOD ref1->method EXPORTING ...
```

Classes in Detail

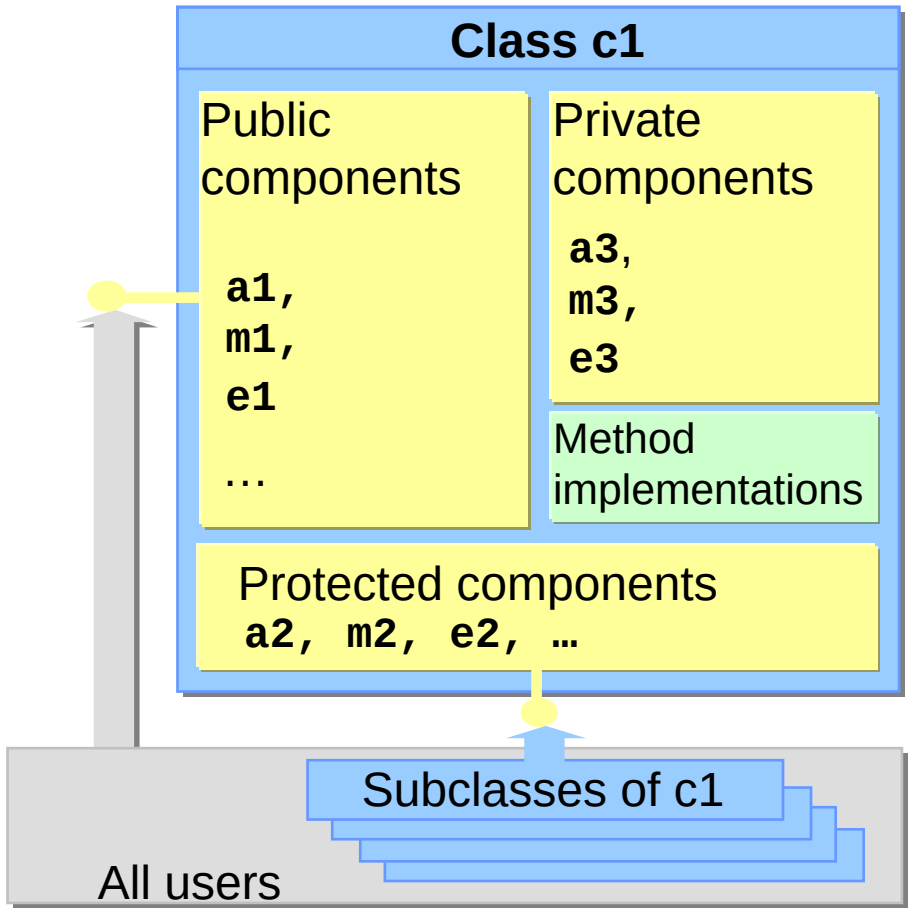


- **Structure of classes**
- **Components of classes**
- **Accessing the components**

Structure of Classes - Visibility Sections

```
CLASS c1 DEFINITION.
  PUBLIC SECTION.
    DATA:    a1 …
    METHODS: m1 …
    EVENTS:  e1 …
  PROTECTED SECTION.
    DATA:    a2 …
    METHODS: m2 …
    EVENTS:  e2 …
  PRIVATE SECTION.
    DATA:    a3 …
    METHODS: m3 …
    EVENTS:  e3 …
ENDCLASS.
```

```
CLASS c1 IMPLEMENTATION.

  METHOD m1. … ENDMETHOD.

  METHOD m2. … ENDMETHOD.

  METHOD m3. … ENDMETHOD.

ENDCLASS.
```



Components of Classes: Attributes

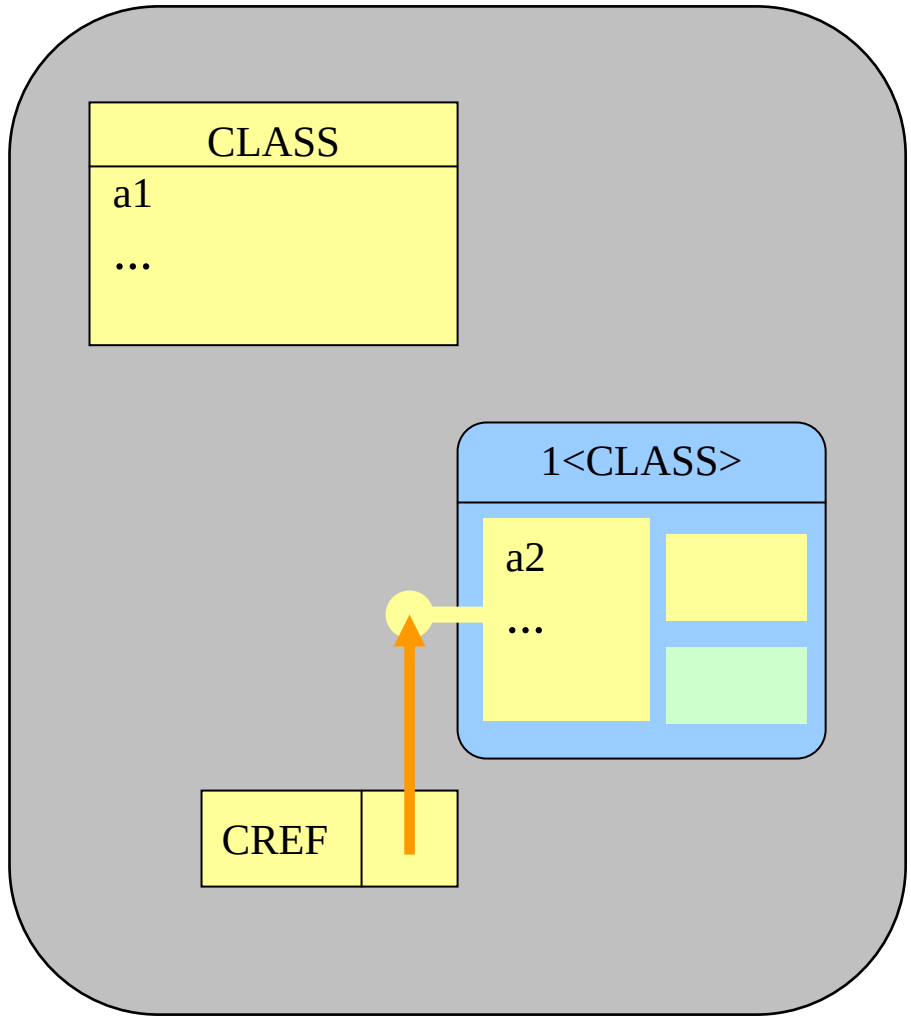
```
CLASS ... DEFINITION.
  ...
  ... SECTION.
    DATA       ... TYPE ... [READ-ONLY] ...
    CLASS-DATA ... TYPE ... [READ-ONLY] ...
    CONSTANTS  ... TYPE ...  VALUE ...
    ...

ENDCLASS.
```

- **DATA:** Instance attributes
- **CLASS-DATA:** Static attributes
- **CONSTANTS:** Constants

Static Attributes and Instance Attributes

```
CLASS c DEFINITION.
  PUBLIC SECTION.
    ...
    CLASS-DATA a1(10) TYPE C
               VALUE 'Static'.
    DATA       a2(10) TYPE C
               VALUE 'Instance'.
    ...
ENDCLASS.
```

```
DATA: cref TYPE REF TO c.


WRITE c=>a1.

CREATE OBJECT cref TYPE c.

WRITE cref->a2.
```



Components of Classes: Methods

```
CLASS ... DEFINITION.
  ...
  ... SECTION.
    METHODS ... IMPORTING  [VALUE] ... TYPE ... [OPTIONAL]
                EXPORTING  [VALUE] ... TYPE ...
                CHANGING   [VALUE] ... TYPE ... [OPTIONAL]
                RETURNING   VALUE(...) TYPE ...
                EXCEPTIONS ...
    CLASS-METHODS ...
    ...
ENDCLASS.
```

```
CLASS ... IMPLEMENTATION.
  METHOD ...
    ...
  ENDMETHOD.
ENDCLASS.
```

- **METHODS:** Instance methods
- **CLASS-METHODS:** Static methods

Constructors

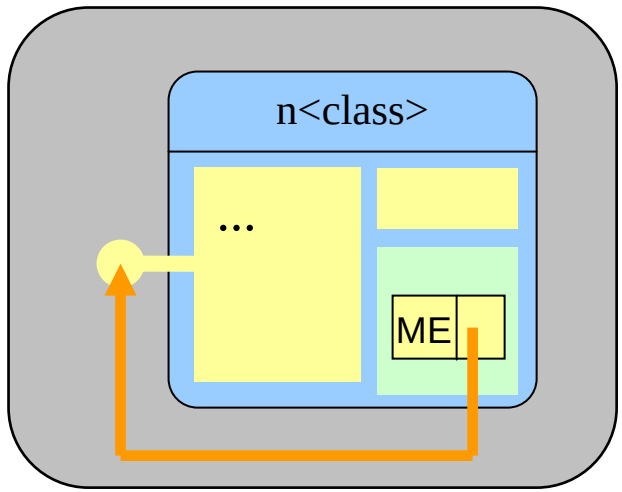
```
CLASS c DEFINITION.
  PUBLIC SECTION.
    METHODS CONSTRUCTOR
            [IMPORTING arg1 TYPE type ... ].

    CLASS-METHODS CLASS_CONSTRUCTOR.

ENDCLASS.
```

```
CLASS c IMPLEMENTATION.
  METHOD CONSTRUCTOR.
    ...
  ENDMETHOD.
  METHOD CLASS_CONSTRUCTOR.
    ...
  ENDMETHOD.
ENDCLASS.
```

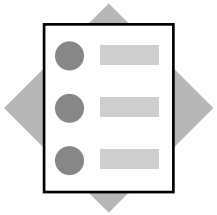
```
PROGRAM … .

DATA o1 TYPE REF TO c.
CREATE OBJECT o1 EXPORTING arg1 = v1 ...
```

Accessing the components of classes

- **Instance components** **ref–>comp**
  - Instance attribute ref->attr
  - Instance method: call method ref->meth
- **Static components** **class=>comp**
  - Static attribute: class=>attr
  - Static method: call method class=>meth
- **Special references in methods**
  - Self reference: ME->comp
  - Pseudo reference SUPER->comp



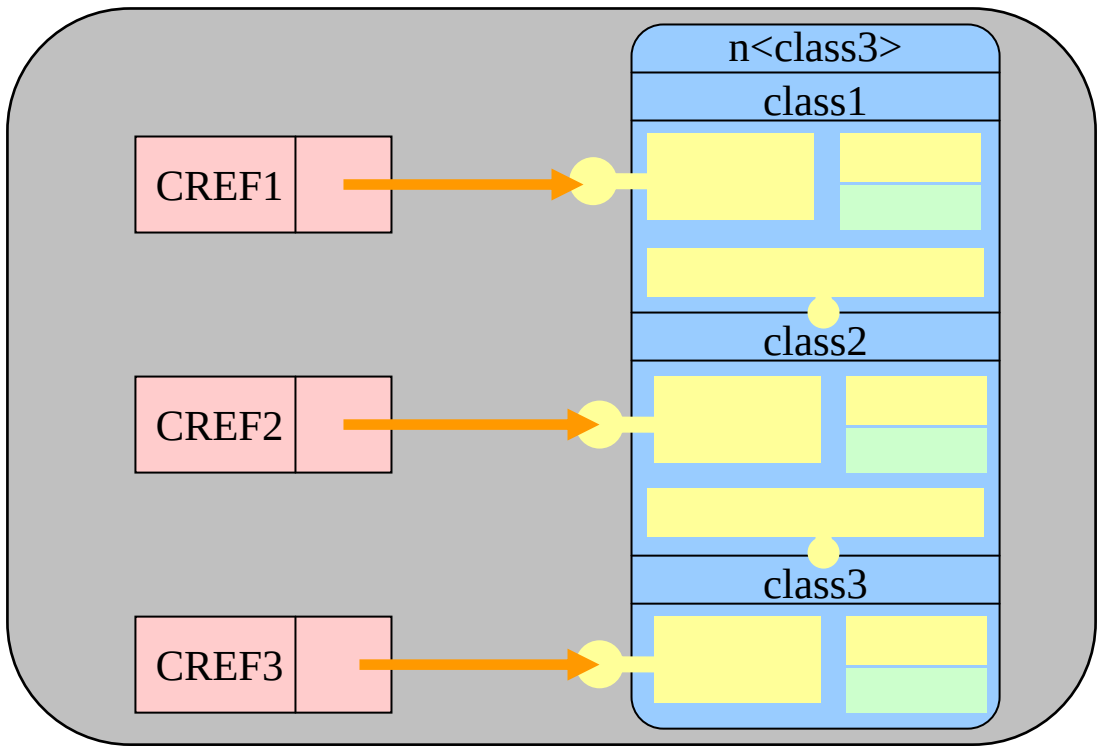
Inheritance



- Introduction
- Overview
- Single inheritance
- Redefining methods
- Example: Subclass of superclass counter

Inheritance: Introduction

- Definition of a class by inheriting the components from a superclass (Reuse)
- Specialization by adding own components and redefining methods in subclasses
- Polymorphism by accessing subclass objects



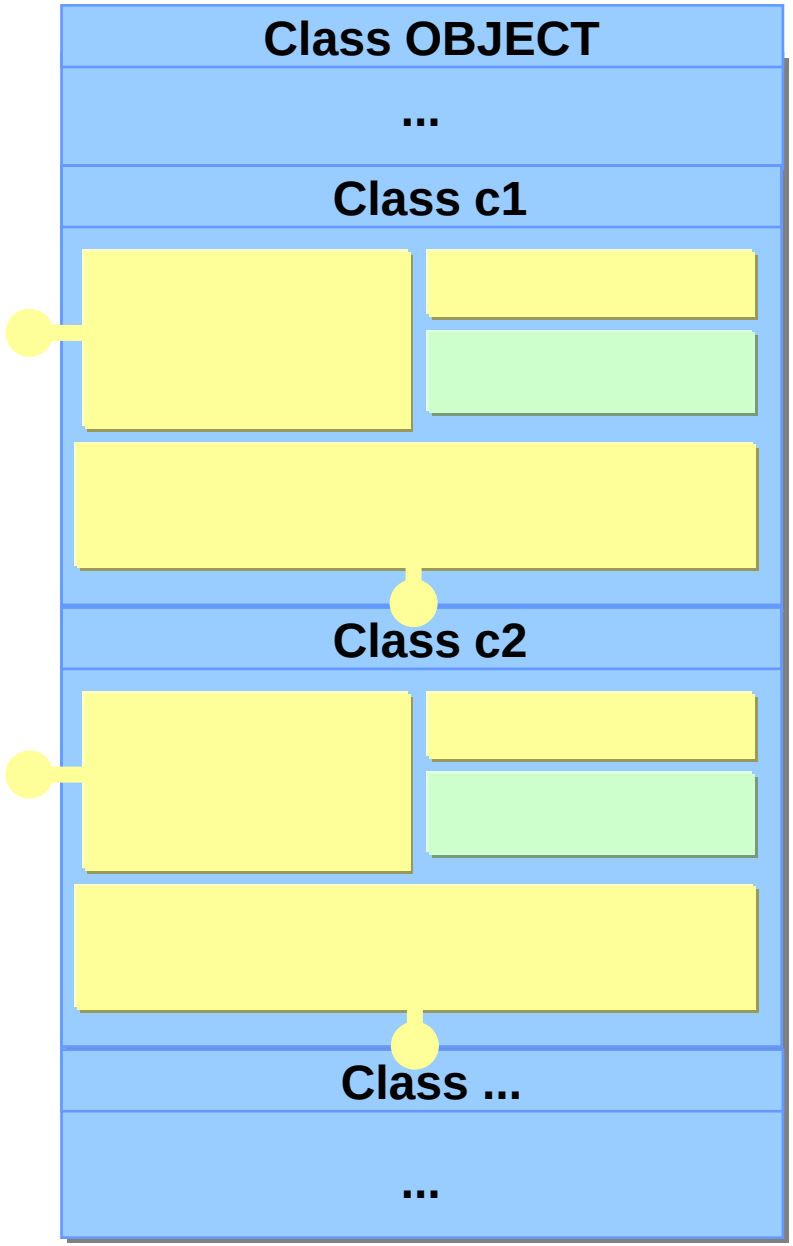
Inheritance - Overview

```
CLASS c1 DEFINITION INHERITING FROM ...
 ...
ENDCLASS.
```

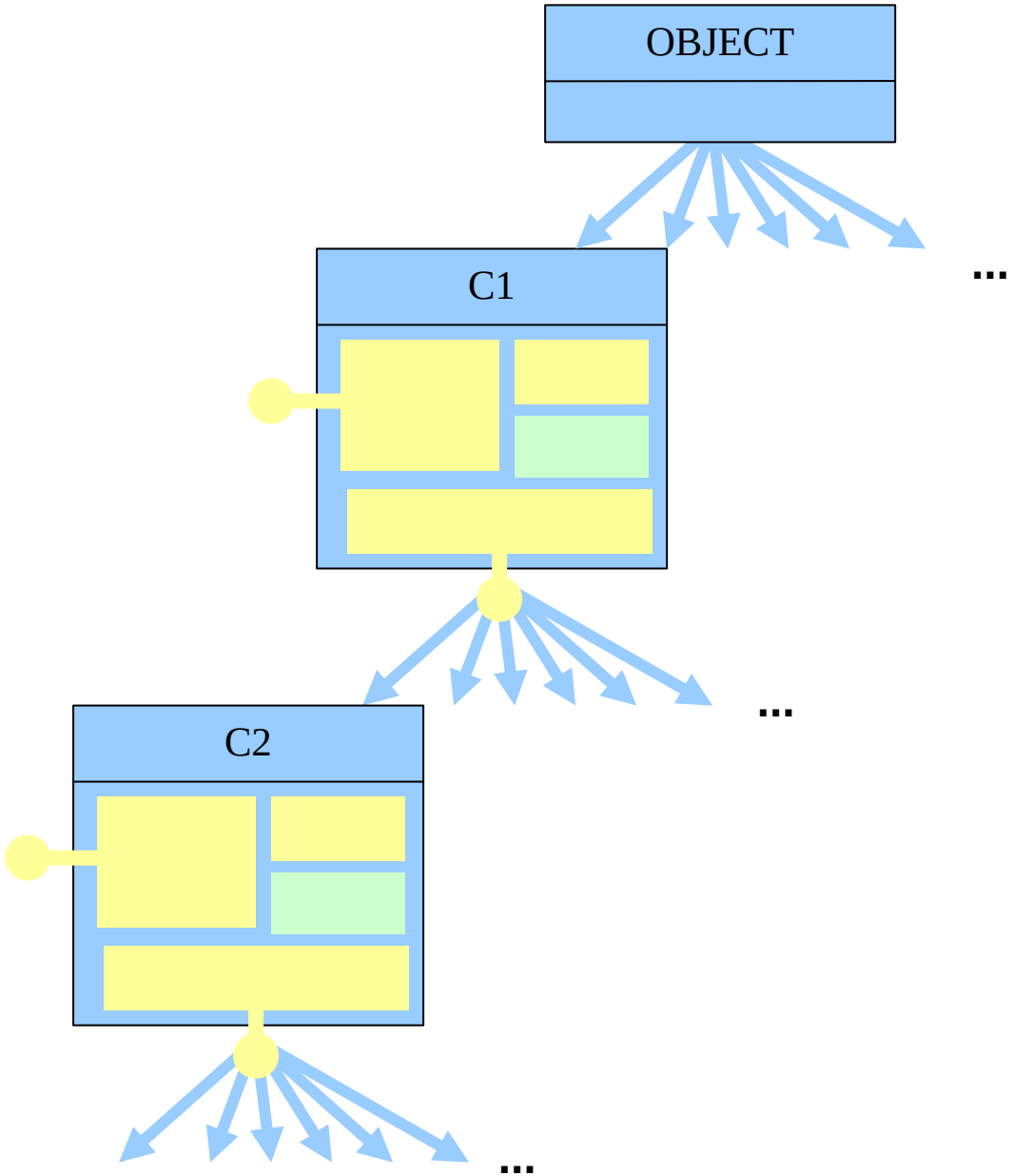
```
CLASS c1 IMPLEMENTATION.
  ...
ENDCLASS.
```

```
CLASS c2 DEFINITION INHERITING FROM c1.
 ...
ENDCLASS.
```

```
CLASS c2 IMPLEMENTATION.
  ...
ENDCLASS.
```



Single Inheritance



Redefining Methods

```
CLASS ... DEFINITION INHERITING FROM ...

  ... SECTION.
    METHODS ... REDEFINITON ...
    ...
  ...

ENDCLASS.
```

```
CLASS ... IMPLEMENTATION.

  METHOD ...
    ...
  ENDMETHOD.

ENDCLASS.
```

Semantic rules

- Subclasses must behave just like their superclass for all users of inherited components
- A redefined method must observe the original semantics
- Inheritance should only be used to specialize

Inheritance and Constructors

```
CLASS subclass DEFINITION INHERITING FROM superclass.
  PUBLIC SECTION.
    METHODS CONSTRUCTOR IMPORTING ...
    ...
ENDCLASS.
```

```
CLASS subclass IMPLEMENTATION.
  METHOD CONSTRUCTOR.
    ...
    CALL METHOD SUPER->CONSTRUCTOR EXPORTING ...
    ...
  ENDMETHOD.
  ...
ENDCLASS.
```

Access to static attributes only

Accress to instance attributes also

```
PROGRAM ...

DATA o1 TYPE REF TO subclass.
CREATE OBJECT o1 TYPE subclass EXPORTING ...
```

Subclass for Superclass Counter*

```
CLASS counter_ten DEFINITION INHERITING FROM counter.
  PUBLIC SECTION.
    METHODS increment REDEFINITION.
    DATA count_ten.
ENDCLASS.
```

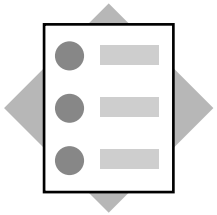
```
CLASS counter_ten IMPLEMENTATION.
   METHOD increment.
     DATA modulo TYPE I.
     CALL METHOD super->increment.
     modulo = count mod 10.
     IF modulo = 0.
       count_ten = count_ten + 1.
     ENDIF.
   ENDMETHOD.
 ENDCLASS.
```

*Replace PRIVATE with:

```
PROTECTED SECTION.
  DATA count TYPE I.
```

```
DATA: count TYPE REF TO counter.

CREATE OBJECT count TYPE counter_ten.
CALL METHOD count->set EXPORTING set_value = number.
DO 10 TIMES.
  CALL METHOD count->increment.
ENDDO.
```

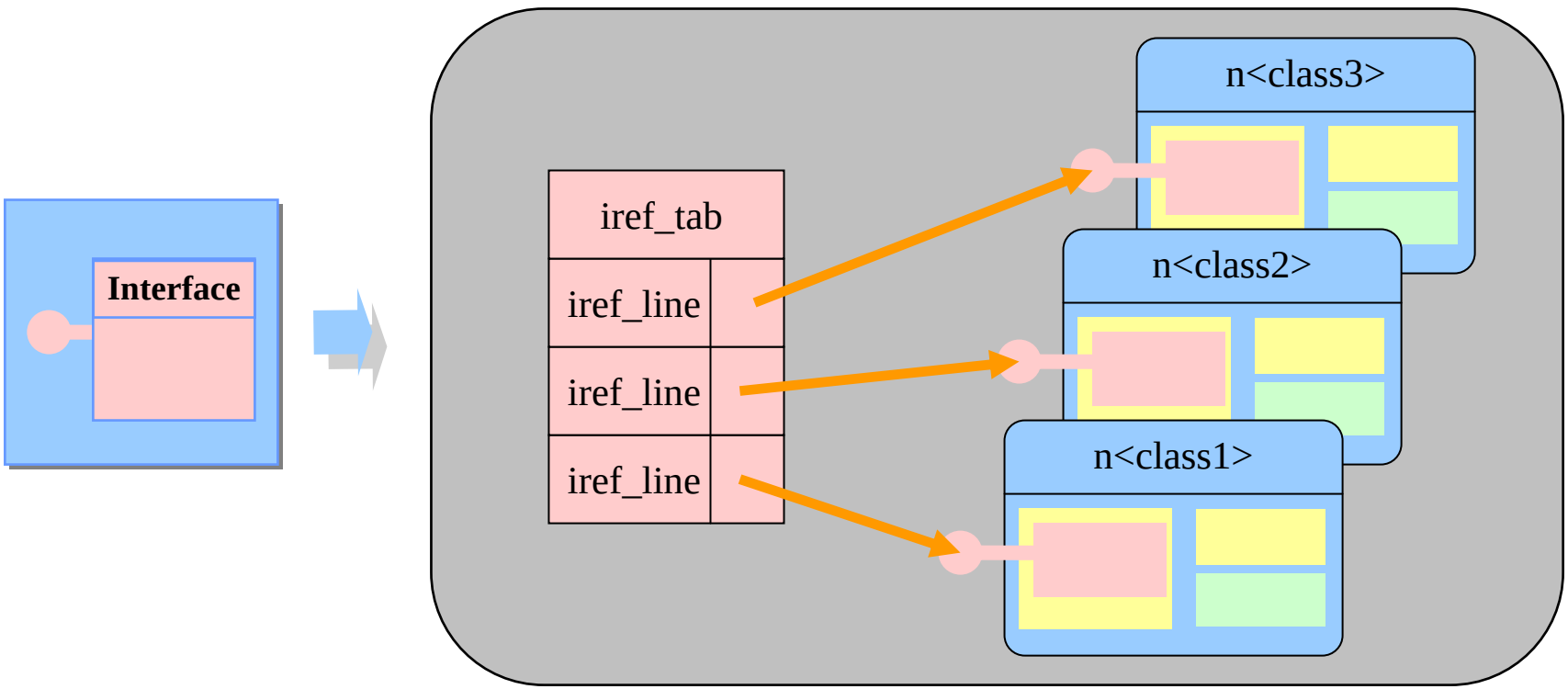
Interfaces



- Introduction
- Overview
- Definition
- Implementation
- Interface references
- Example: Interface for counter

Interfaces: Introduction

- Definition of an interface without implementation
- Classes can implement several interfaces
- Uniform access with interface references
- Polymorphism independent from inheritance

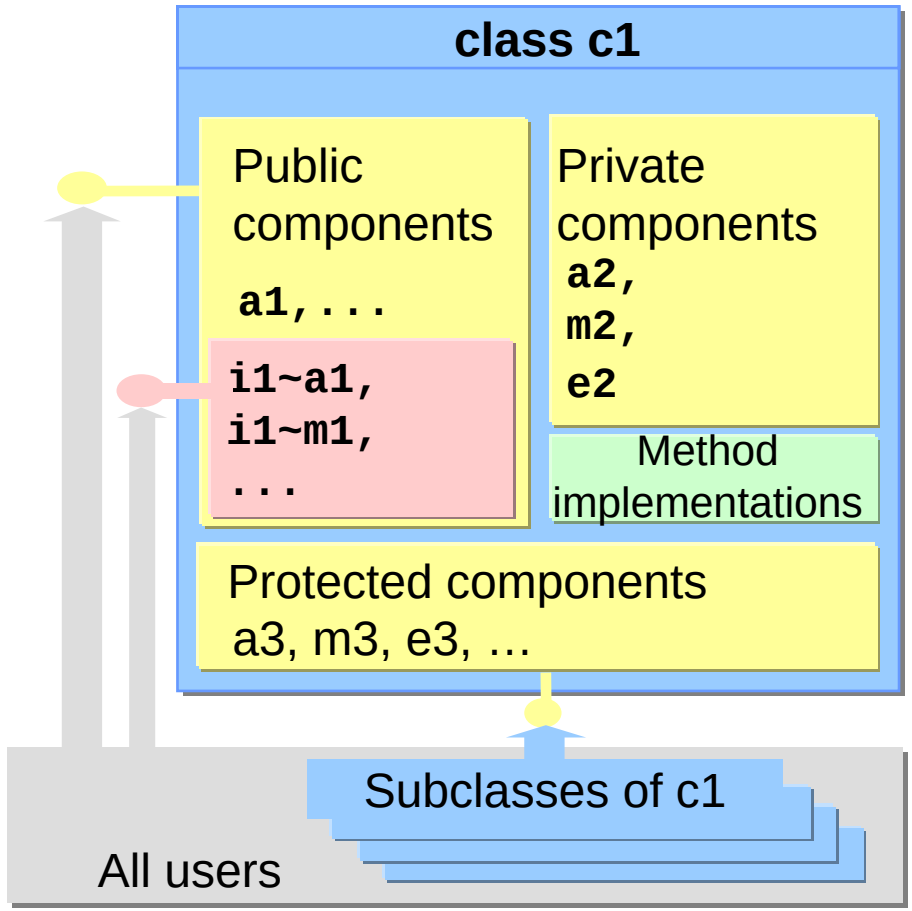


Interfaces - Overview

```
INTERFACE i1.
  DATA:    a1 ...
  METHODS: m1 ...
  EVENTS:  e1 ...
ENDINTERFACE.
```

```
CLASS c1 DEFINITION.
  PUBLIC SECTION.
    DATA       a1 ...
    INTERFACES i1 ...
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.
```

```
CLASS c1 IMPLEMENTATION.
METHOD i1~m1. … ENDMETHOD.
ENDCLASS.
```



Interfaces - Definition

```
INTERFACE ... .
  ...
  DATA:        ... TYPE ... [READ-ONLY] ...
  CLASS-DATA: ... TYPE ... [READ-ONLY] ...
  CONSTANTS:  ... TYPE ... [VALUE ...]

  METHODS: ... IMPORTING  [VALUE] ... TYPE ... [OPTIONAL]
               EXPORTING  [VALUE] ... TYPE ...
               CHANGING   [VALUE] ... TYPE ... [OPTIONAL]
               RETURNING   VALUE(...) TYPE ...
               EXCEPTIONS ...
  CLASS-METHODS: ...

  EVENTS: ... [EXPORTING VALUE(...) TYPE ... [OPTIONAL]].
  CLASS-EVENTS:...

  INTERFACES: ...

ENDINTERFACE.
```

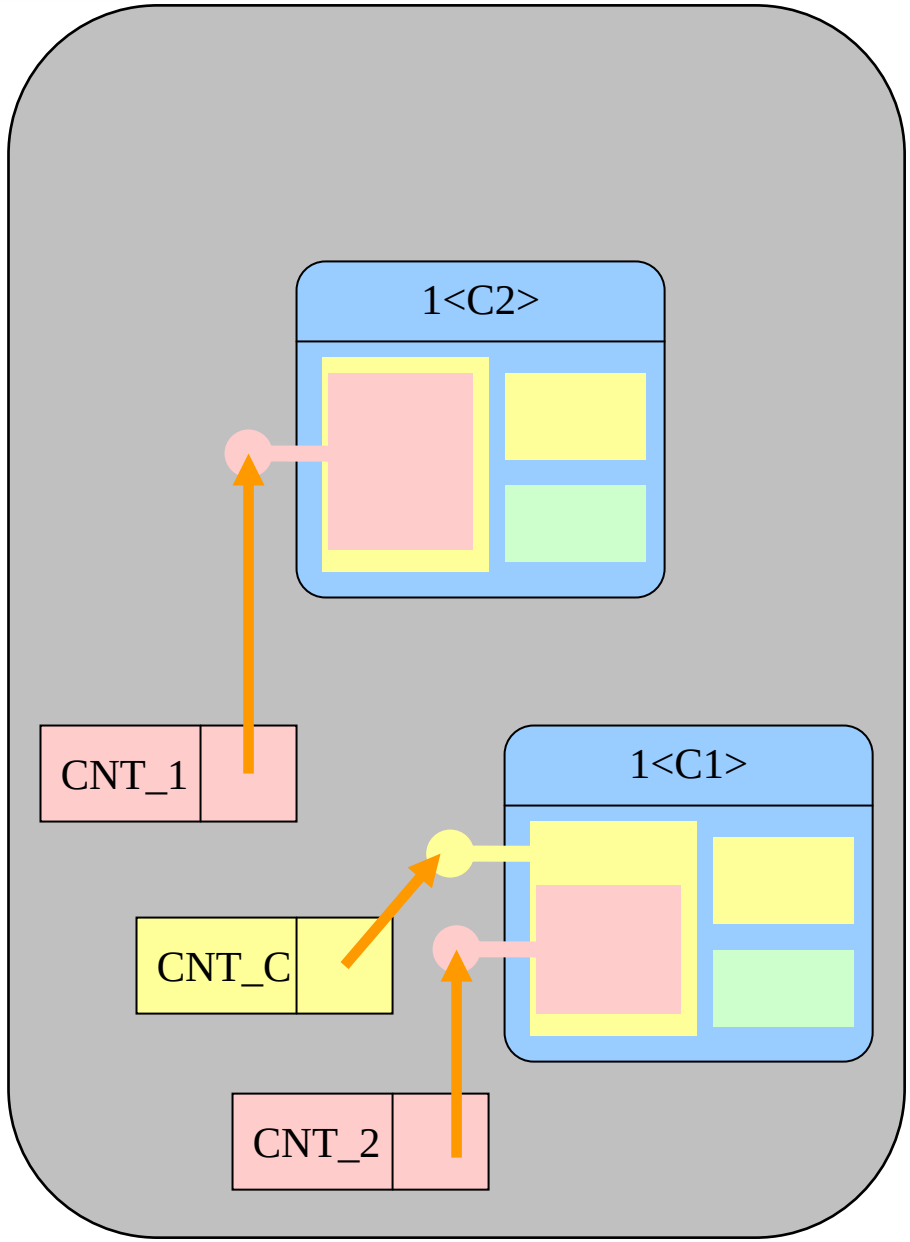
Interfaces - Implementation

```
CLASS ... DEFINITION.

  PUBLIC SECTION.
    INTERFACES: ...
    ...
  ...

ENDCLASS.
```

```
CLASS ... IMPLEMENTATION.

  METHOD ...~...
    ...
  ENDMETHOD.

ENDCLASS.
```

Interface References

```
INTERFACE i1.
  ...
ENDINTERFACE.

CLASS c1 DEFINITION.
  PUBLIC SECTION.
    DATA a1.
    INTERFACES i1.
ENDCLASS.

CLASS c2 DEFINITION.
  PUBLIC SECTION.
    INTERFACES i1.
ENDCLASS.

DATA  cnt_c TYPE REF TO c1.
DATA: cnt_1 TYPE REF TO i1,
      cnt_2 LIKE cnt_1.

CREATE OBJECT: cnt_c TYPE c1,
               cnt_1 TYPE c2.

MOVE cnt_c to cnt_2.
```



Interface for Object State - Implementation

```
INTERFACE status.
  METHODS write.
ENDINTERFACE.
```

```
CLASS counter DEFINITION.
  PUBLIC SECTION.
    INTERFACES status.
    METHODS increment.
  PRIVATE SECTION.
    DATA count TYPE i.
ENDCLASS.
```

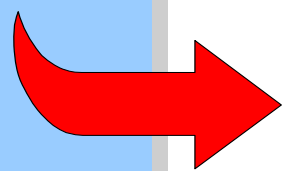
```
CLASS counter IMPLEMENTATION.
  METHOD status~write.
  WRITE: 'Count in counter is',
           count.
  ENDMETHOD.
  METHOD increment.
    count = count + 1.
  ENDMETHOD.
ENDCLASS.
```

```
CLASS bicycle DEFINITION.
  PUBLIC SECTION.
    INTERFACES status.
    METHODS drive.
  PRIVATE SECTION.
    DATA speed TYPE i.
 ENDCLASS.
```

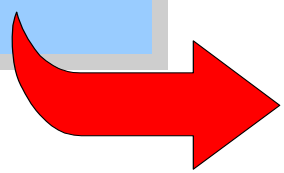
```
CLASS bicycle IMPLEMENTATION.
  METHOD status~write.
  WRITE: 'Speed of bicycle is',
           speed.
  ENDMETHOD.
  METHOD drive.
    speed = speed + 10.
  ENDMETHOD.
ENDCLASS.
```

Interface for Object State - Use

```
DATA: count  TYPE REF TO counter,
      bike   TYPE REF TO bicycle,
      status TYPE REF TO status.

CREATE OBJECT: count, bike.

DO 5 TIMES.
  CALL METHOD: count->increment,
               bike->drive.
ENDDO.

status = count.
CALL METHOD status->write.

status = bike.
CALL METHOD status->write.
```

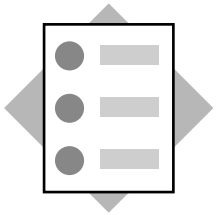


Counter status



Bike speed

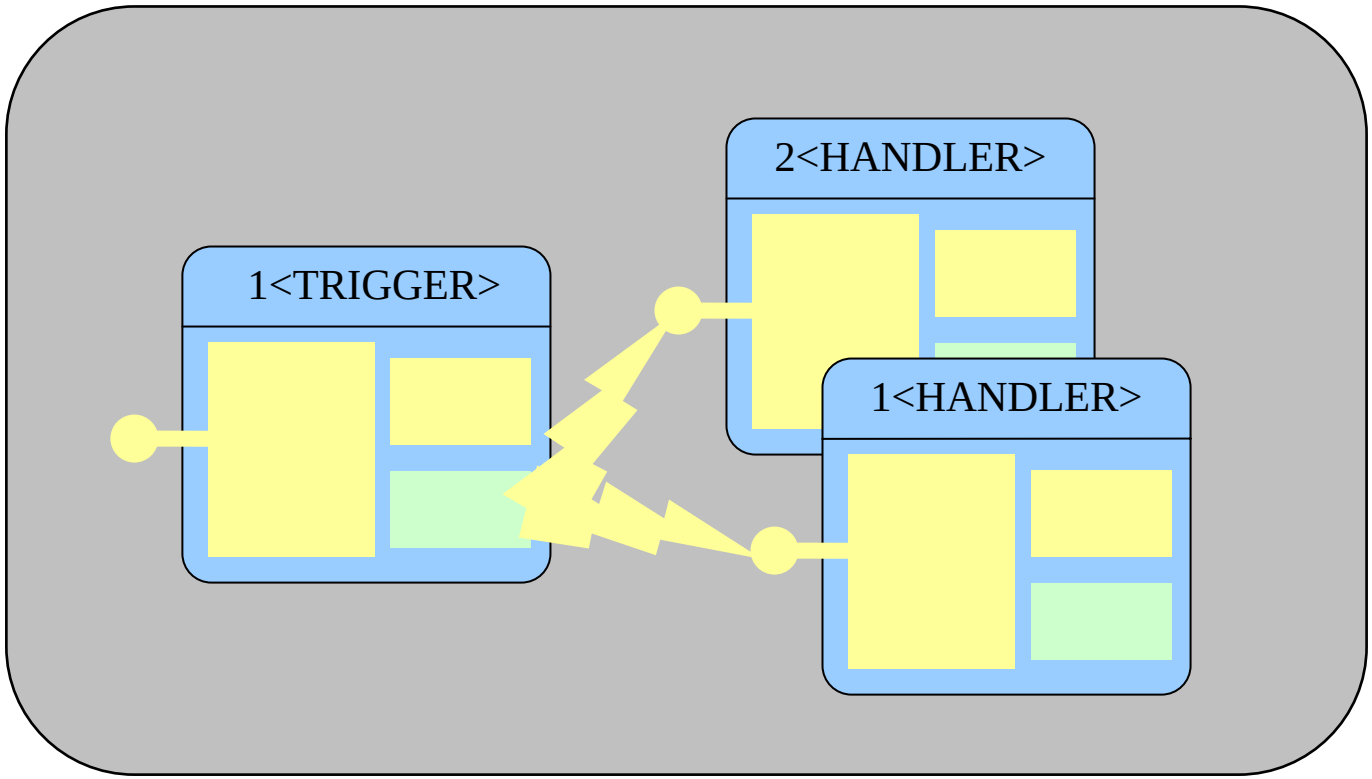
Events



- Introduction
- Overview
- Declaring and triggering events
- Event handler methods
- Handling events
- Example: Overflow in counter

Events - Introduction

- Events are components of classes
- Methods can raise the events of their class
- Handler methods can be triggered by events



Events - Overview

```
CLASS c1 DEFINITION.
 PUBLIC SECTION.
   EVENTS e1 EXPORTING VALUE(p1)
                       TYPE i.
   METHODS m1.

 PRIVATE SECTION.
   DATA a1 TYPE i.

ENDCLASS.
```

```
CLASS c1 IMPLEMENTATION.
  METHOD m1.
    a1 = ...
    RAISE EVENT e1
          EXPORTING p1 = a1.
  ENDMETHOD.
ENDCLASS.
```

Event trigger

```
CLASS c2 DEFINITION.
 PUBLIC SECTION.
   METHODS m2 FOR EVENT e1 OF c1
          IMPORTING p1.


 PRIVATE SECTION.
   DATA a2 TYPE i.

ENDCLASS.
```

```
CLASS C2 IMPLEMENTATION.
  METHOD m2.
    a2 = p1.
    ...
  ENDMETHOD.
ENDCLASS.
```

Event handler

Declaring and Triggering Events

```
CLASS ... DEFINITION.

  ... SECTION.

    METHODS ...

    EVENTS ... [EXPORTING VALUE(...) TYPE ... [OPTIONAL]].

    CLASS-EVENTS ...

ENDCLASS.
```

```
CLASS ... IMPLEMENTATION.

  METHOD ...
    ...
    RAISE EVENT ... EXPORTING ... = ...
    ...
  ENDMETHOD.

ENDCLASS.
```

Event Handler Methods

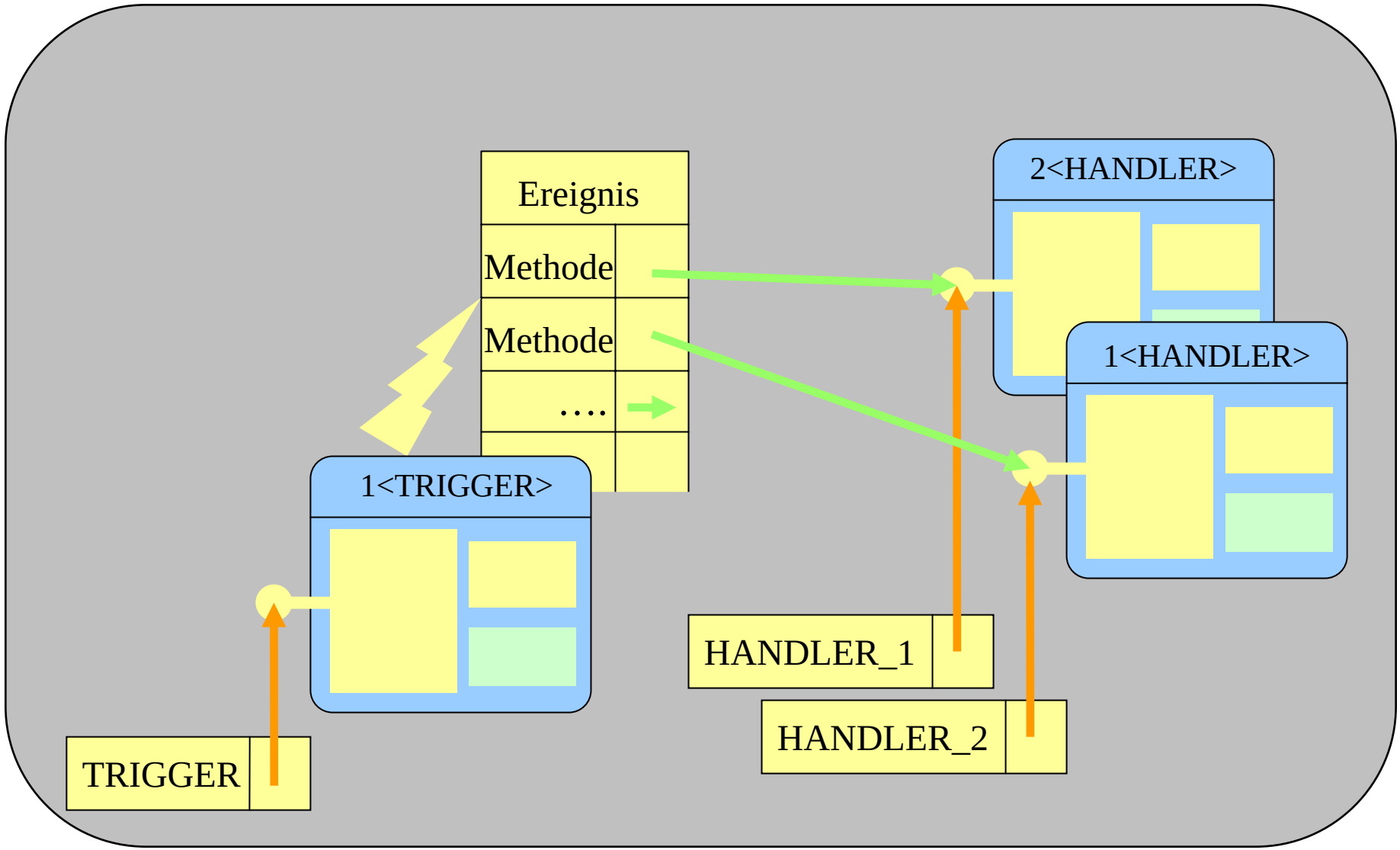
```
CLASS ... DEFINITION.

  ... SECTION.

    METHODS ... FOR EVENT ... OF ... [IMPORTING ... SENDER ... ].


ENDCLASS.
```

```
CLASS ... IMPLEMENTATION.

  METHOD ...
    ...
  ENDMETHOD.

ENDCLASS.
```

Event Handling - Registration

```
PROGRAM ...

DATA: trigger   TYPE REF TO trigger,
      handler_1 TYPE REF TO handler,
      handler_2 TYPE REF TO handler.

CREATE OBJECT: trigger, handler_1, handler_2.

SET HANDLER handler_1->handle_event
            handler_2->handle_event FOR trigger.

CALL METHOD trigger->raise_event.
```

Handling Events - References



Threshold in Counter - Trigger

```
CLASS counter DEFINITION.
  PUBLIC SECTION.
    METHODS increment.
    EVENTS  critical_value EXPORTING value(excess) TYPE i.
  PRIVATE SECTION.
    DATA: count     TYPE i,
          threshold TYPE i VALUE 10.
ENDCLASS.
```

```
CLASS counter IMPLEMENTATION.
  METHOD increment.
    DATA diff TYPE i.
      count = count + 1.
    IF count > threshold.
      diff = count - threshold.
      RAISE EVENT critical_value
            EXPORTING excess = diff.
    ENDIF.
  ENDMETHOD.

ENDCLASS.
```

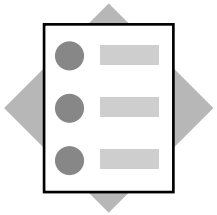
Threshold in Counter - Handling

```
CLASS handler DEFINITION.
  PUBLIC SECTION.
    METHODS handle_excess FOR EVENT critical_value OF
                          counter IMPORTING excess.
ENDCLASS.
```

```
CLASS handler IMPLEMENTATION.
  METHOD handle_excess.
    WRITE: / 'Excess is', excess.
  ENDMETHOD.
ENDCLASS.
```

```
DATA: cnt   TYPE REF TO counter,
      react TYPE REF TO handler.

CREATE OBJECT: cnt, react.

SET HANDLER react->handle_excess FOR ALL INSTANCES.

DO 20 TIMES.
  CALL METHOD cnt->increment.
ENDDO.
```

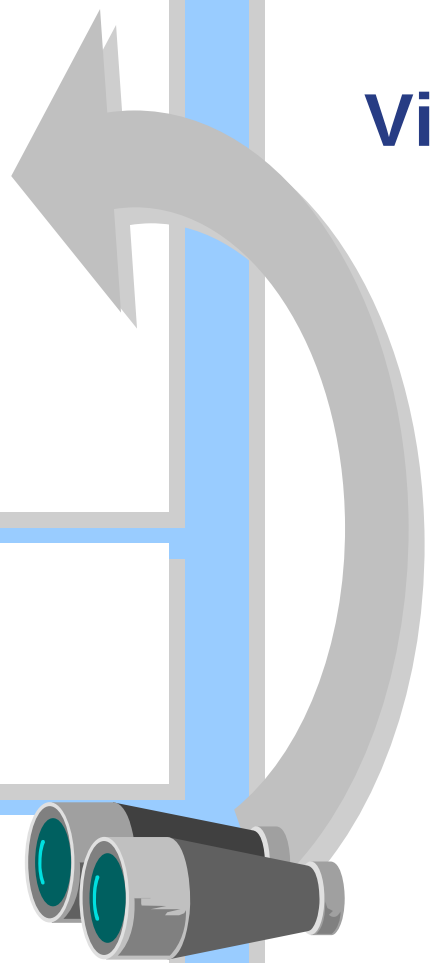
Using Global Classes



- Class pools
- Class Browser and Class Builder
- Class Builder
- Example: Using CL_GUI_PICTURE

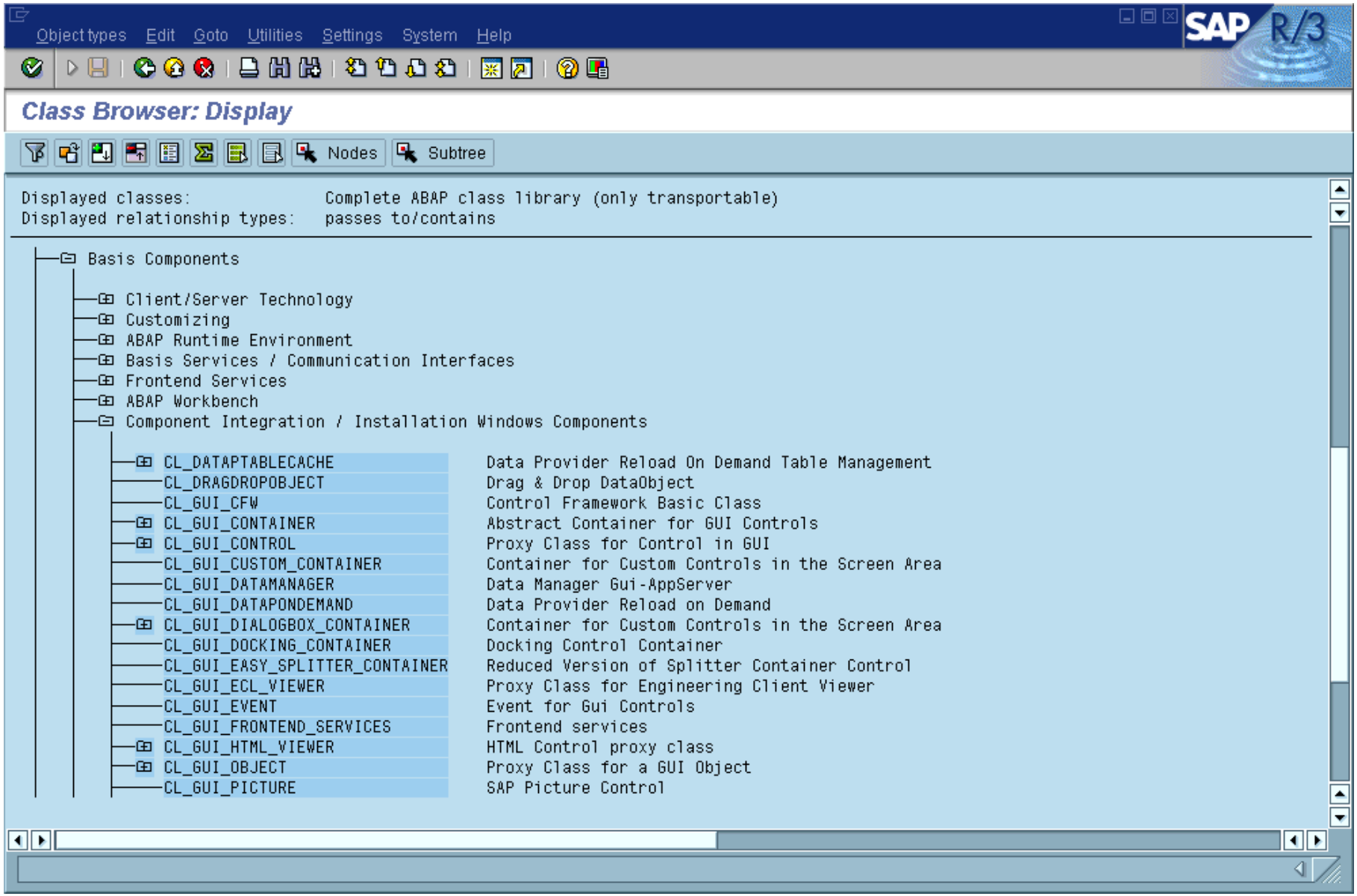
Class Pools

```
CLASS-POOL ...

TYPES …

CLASS …
  ...
ENDCLASS.

INTERFACE …
  ...
ENDINTERFACE.

CLASS … DEFINITION PUBLIC.
  ...
ENDCLASS.

CLASS … IMPLEMENTATION.
  ...
ENDCLASS.
```

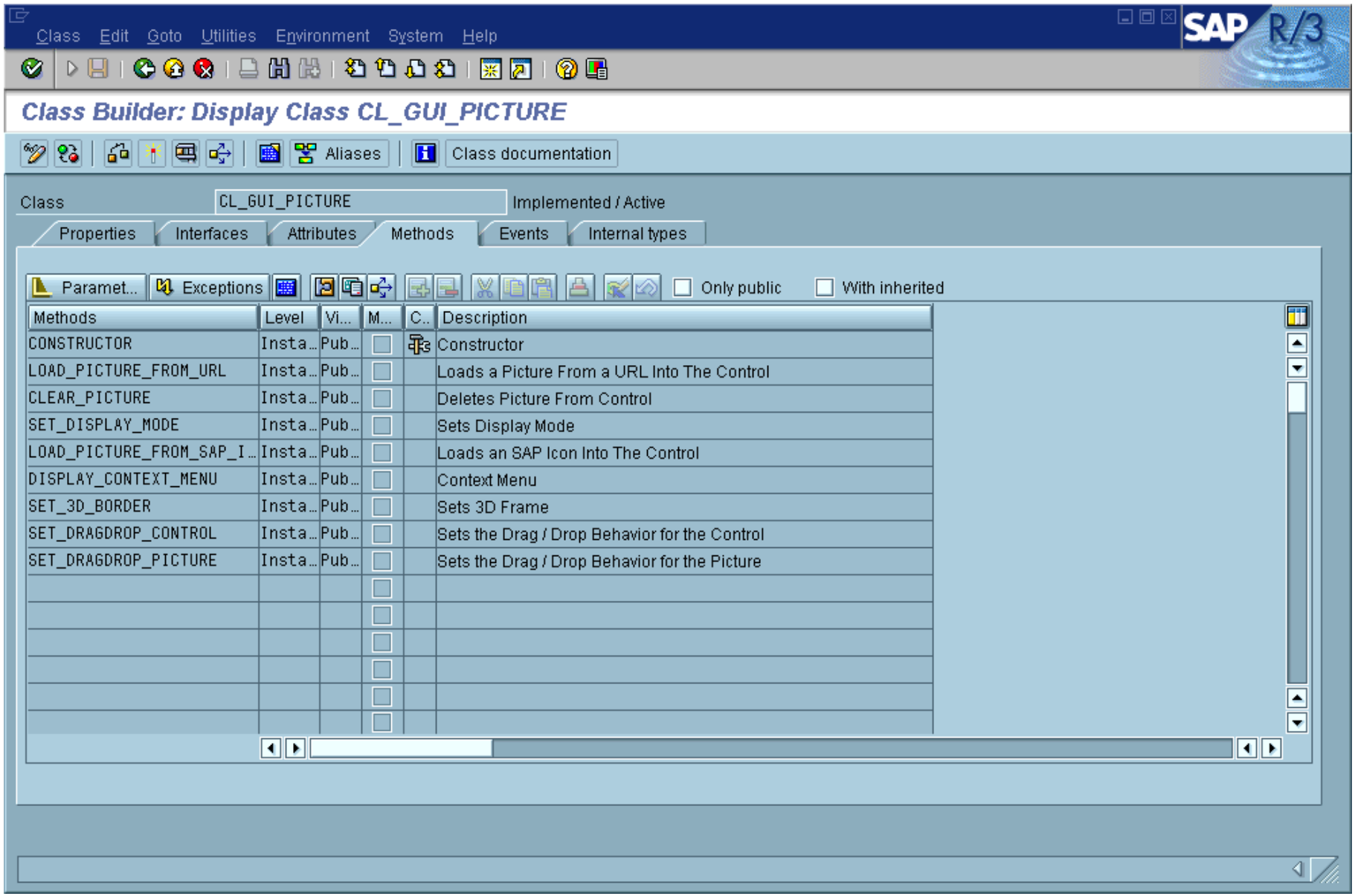


Visibility

Class Library and Class Browser



Class Builder

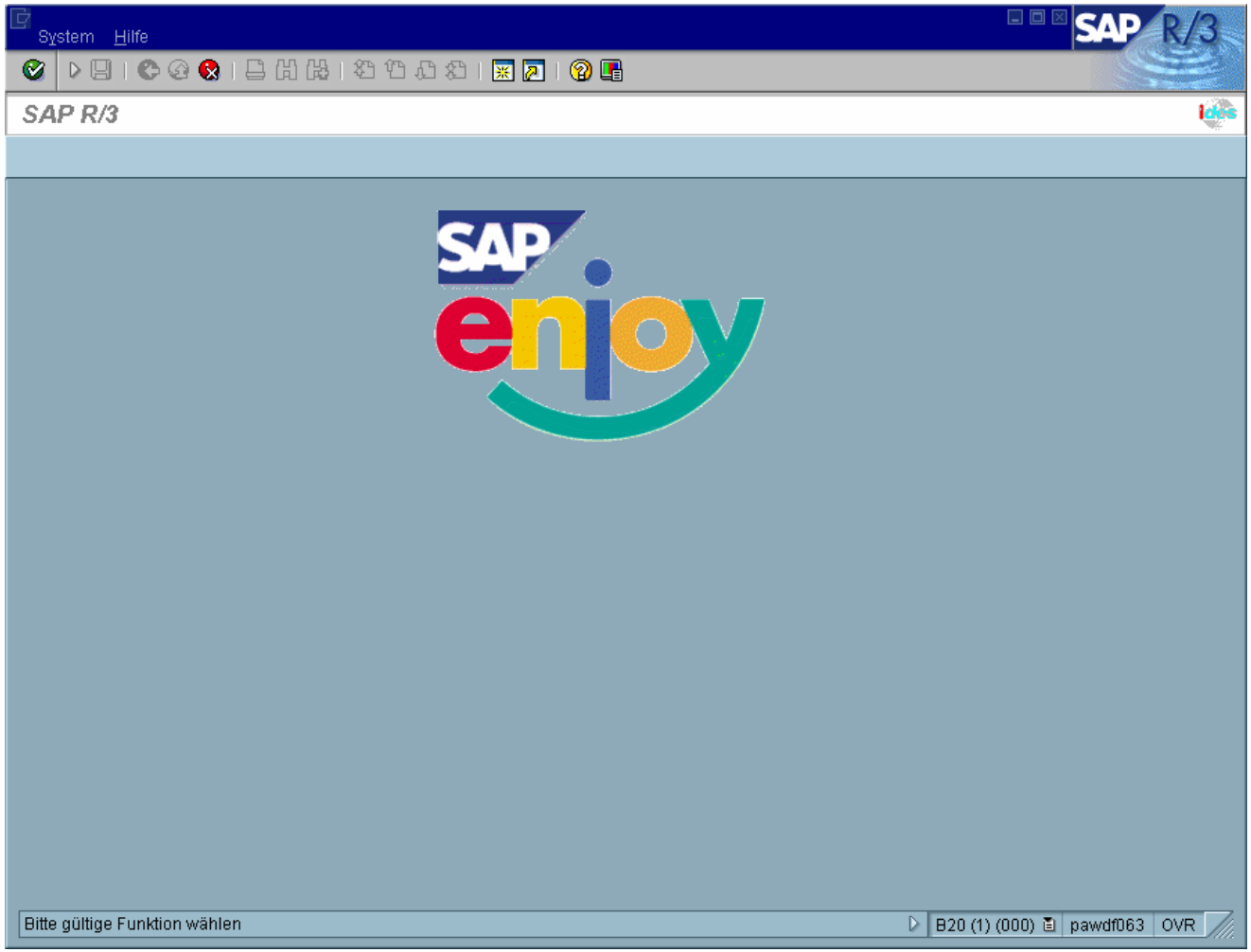


Using a Global Class

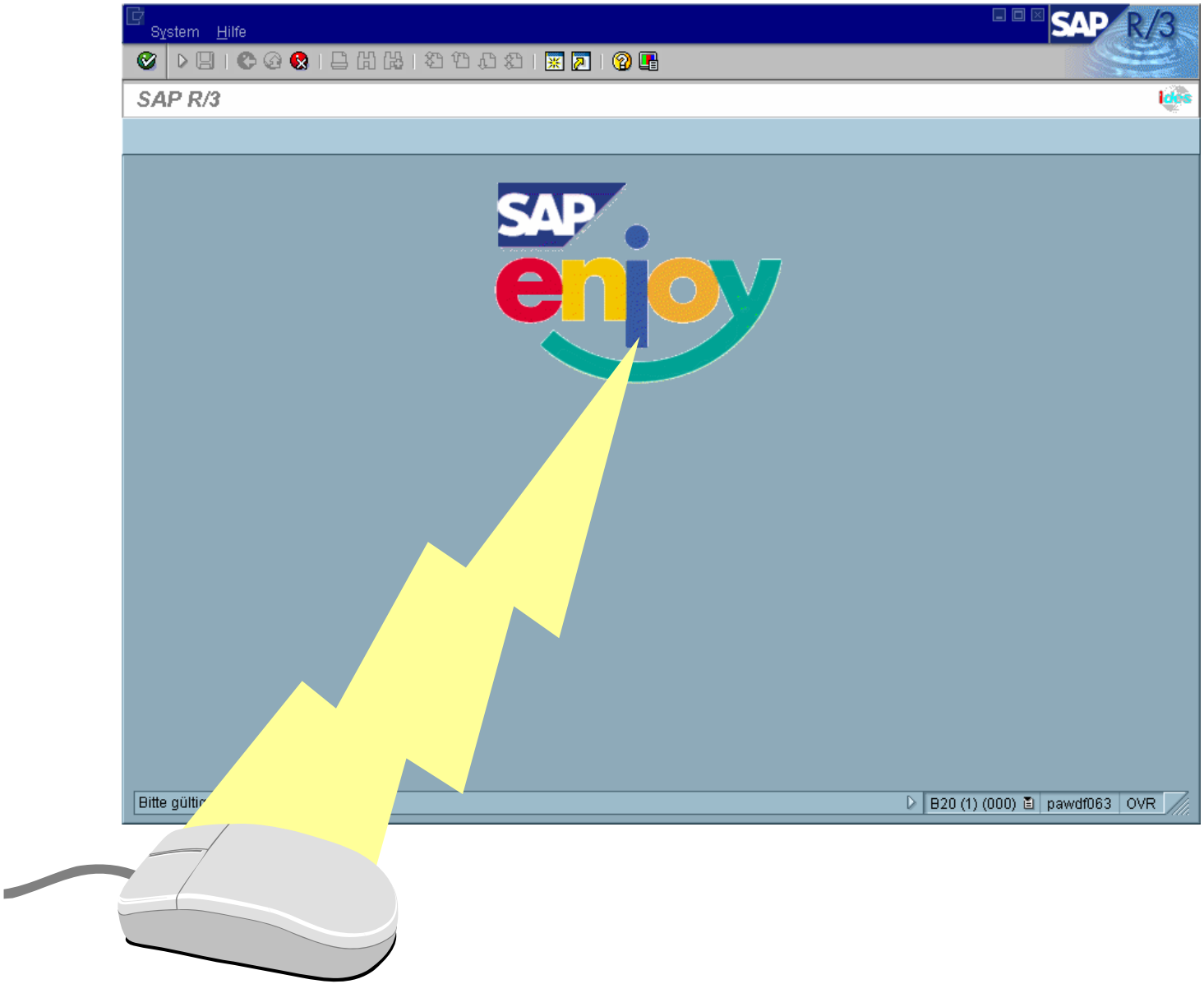
```
CLASS reaction DEFINITION.
 PUBLIC SECTION.
   METHODS on_single_click FOR EVENT picture_click OF cl_gui_picture.
ENDCLASS.

DATA: container1 TYPE REF TO cl_gui_custom_container,
      container2 LIKE container1,
      pict1      TYPE REF TO cl_gui_picture,
      pict2      LIKE pict1,
      react      TYPE REF TO reaction,...

CREATE OBJECT: container1 EXPORTING container_name = 'PICTURE1',
               container2 EXPORTING container_name = 'PICTURE2',
               pict1      EXPORTING parent = container1,
               pict2      EXPORTING parent = container2,
               react.

SET HANDLER react->on_single_click FOR pict1.

CALL METHOD pict1->load_picture_from_url ...
```

```
CLASS reaction IMPLEMENTATION.
  METHOD on_single_click.
   CALL METHOD pict2->load_picture_from_url ...
 ENDMETHOD.
ENDCLASS.
```

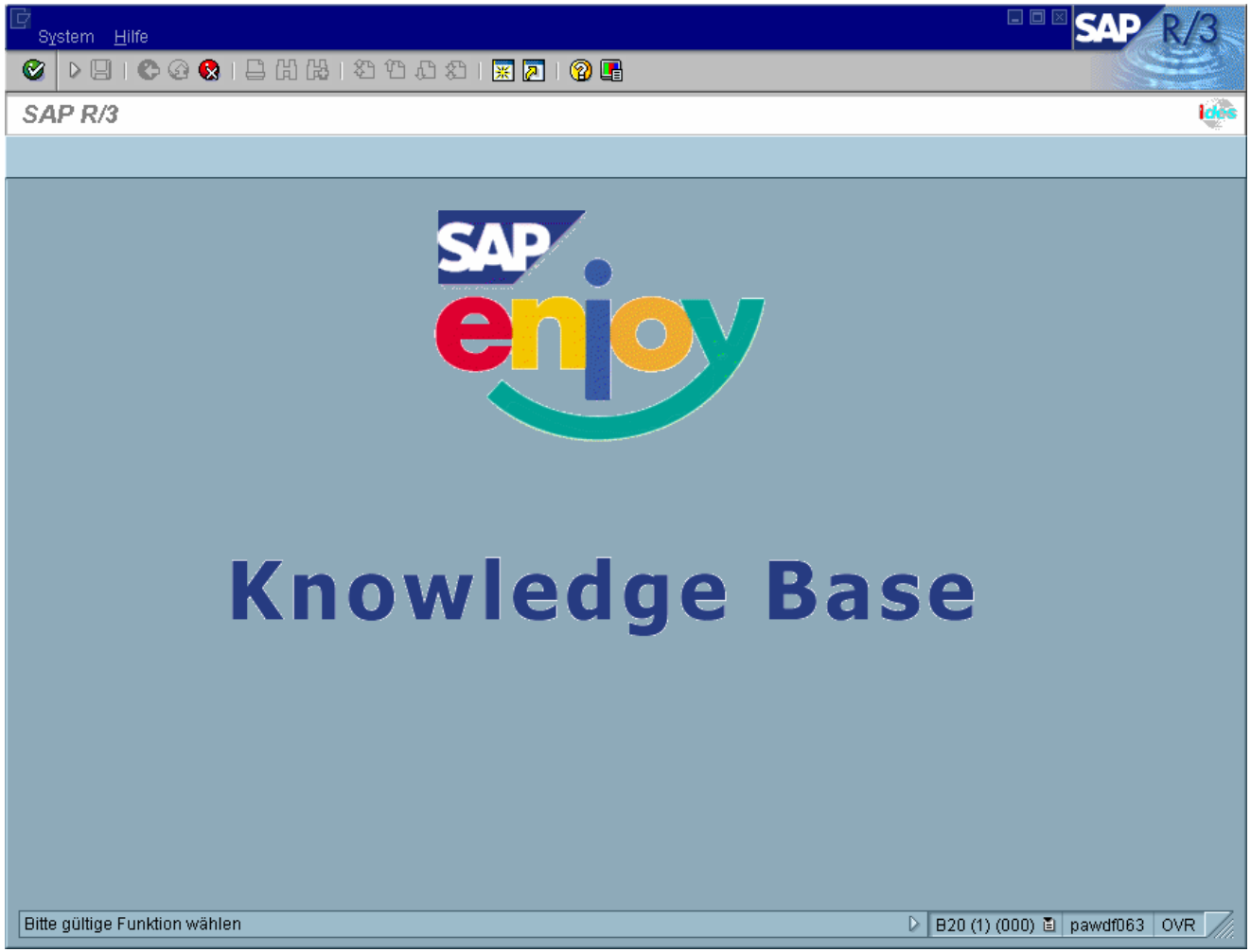
Example of CL_GUI_PICTURE



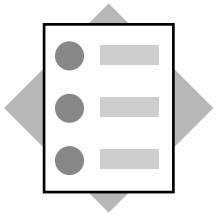
Mouse Click


System Hilfe
SAP R/3
SAP R/3
enjoy
Bitte gülti
B20 (1) (000)
pawdf063
OVR

After Mouse Click



Further Information

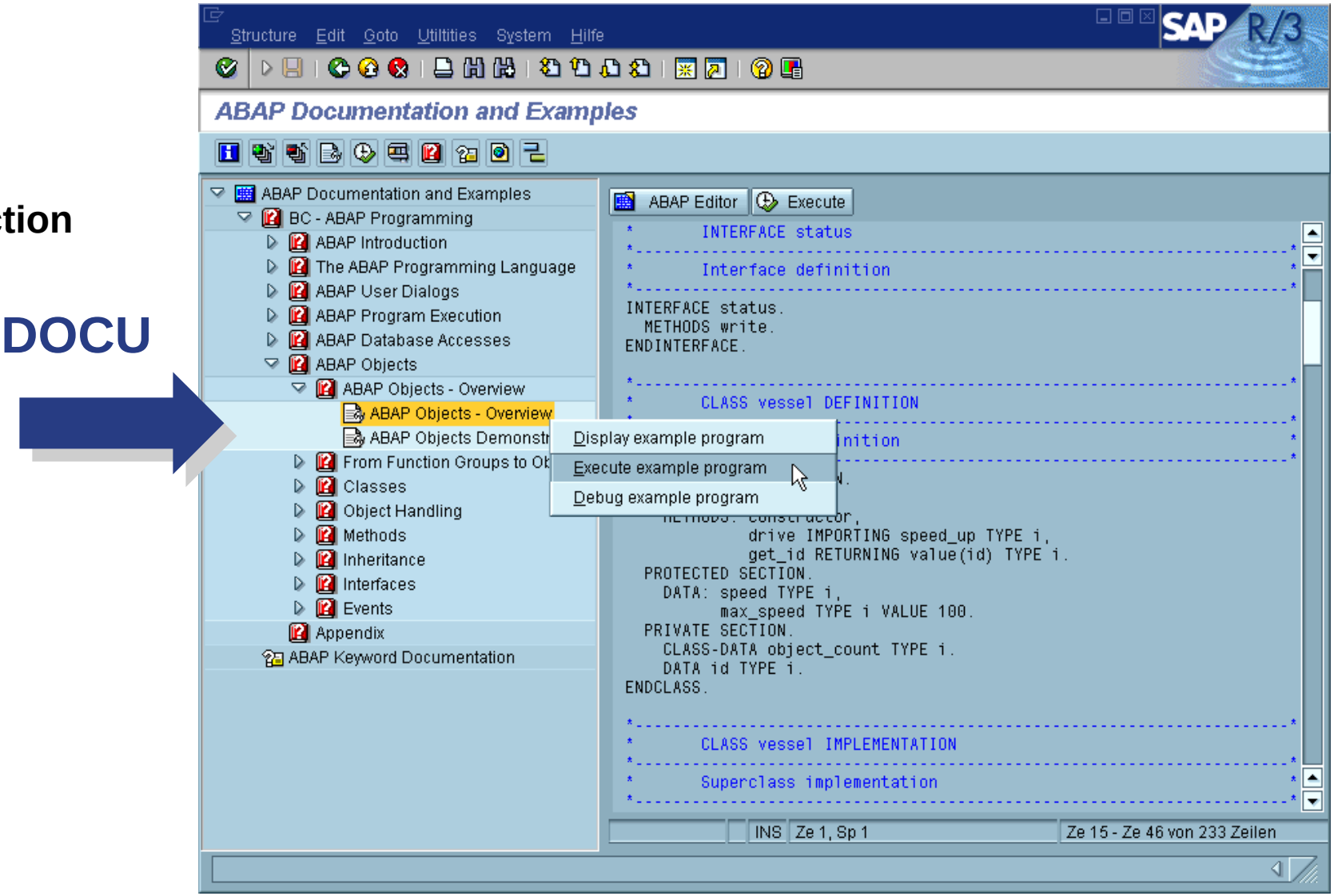


- **Transaction ABAPDOCU**
- **Keyword documentation**

Further Information

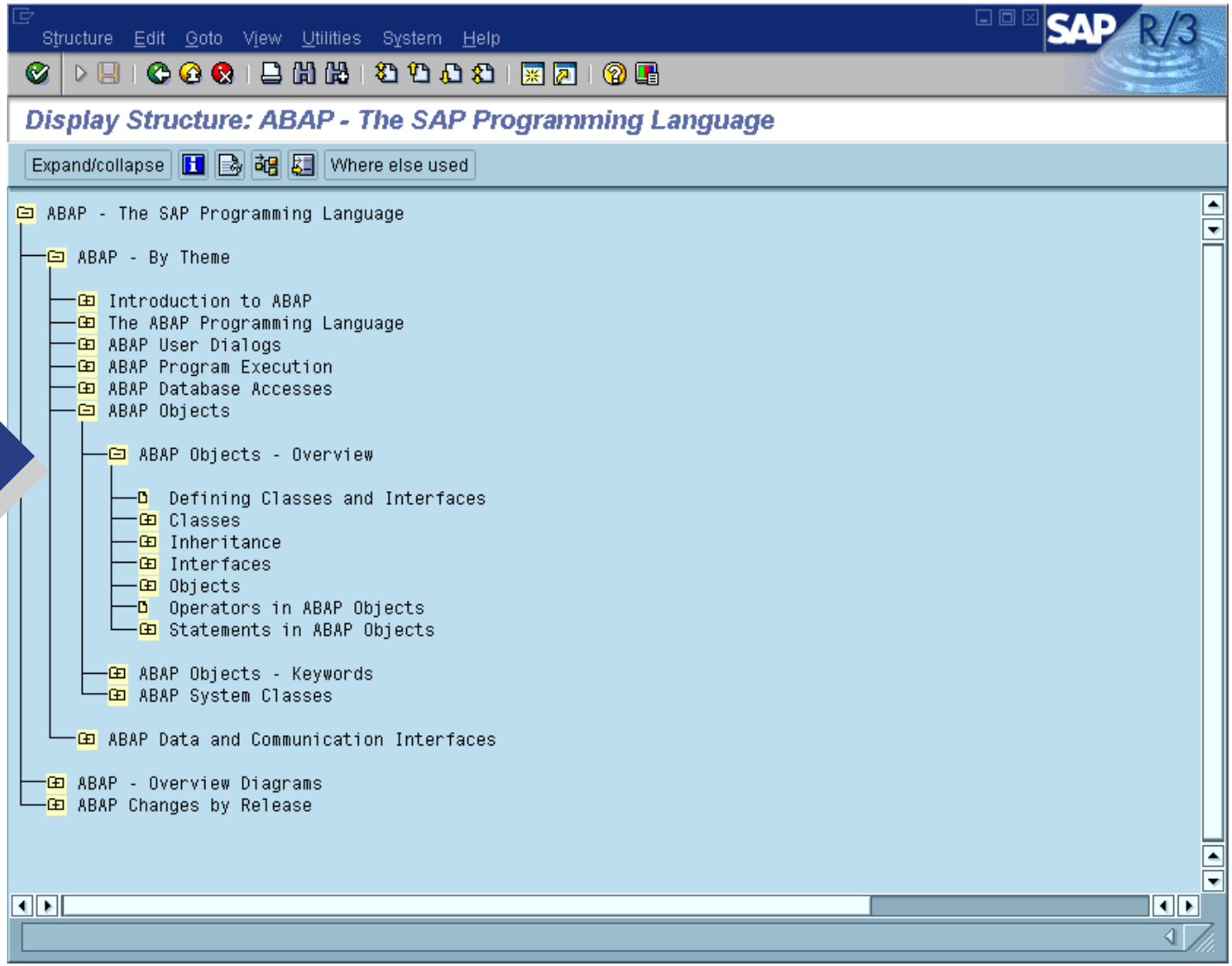
Transaction

ABAPDOCU

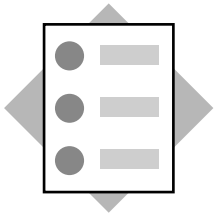


Further Information

Keyword Documentation



Exercises

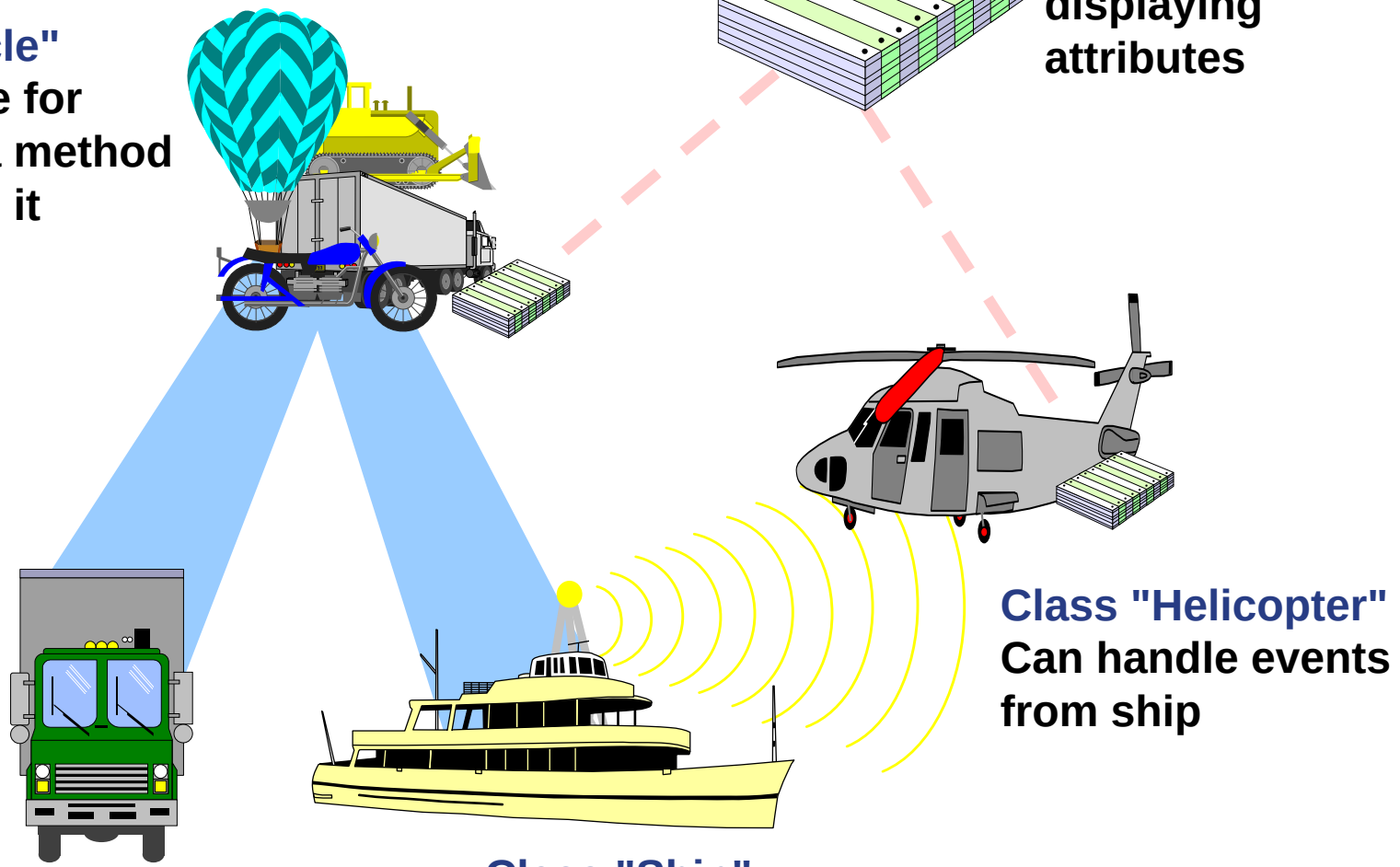


1. Defining classes
2. Creating objects
3. Deriving classes using inheritance
4. Using interfaces
5. Triggering and handling events

Exercises - Scenario

Interface Status
With methods for displaying attributes

Class "Vehicle"
with attribute for speed, and a method for changing it



Class "Helicopter"
Can handle events from ship

Class "Truck"
with its own maximum speed and attribute output

Class "Ship"
with its own maximum speed and attribute output, a name, and an event

Exercise 1 - Classes

- Copy the template ABAP_OBJECTS_ENJOY_0 into an own program.
- Define a local class VEHICLE in the designated area in front of the predefined class MAIN.
- Class VEHICLE should have the protected instance attributes SPEED and MAX_SPEED for its speed and maximum speed, and the public methods SPEED_UP, STOP, and WRITE.
- SPEED_UP should have an IMPORTING parameter STEP. The method should increase the speed by STEP, but not allow it to exceed the maximum speed.
- STOP should reset the speed to zero.
- WRITE should display a list line containing the speed and the maximum speed.

Exercise 2 - Objects

- Continuing the program from exercise 1, create objects from the class VEHICLE.
- Program the respective coding in the predefined method START of class MAIN.
- Define a reference variable VEHICLE with type VEHICLE, and an internal table VEHICLE_TAB, whose line type is also a reference variable to this class.
- In a DO loop, create a number of instances of the class VEHICLE and insert them into the internal table.
- In a LOOP construction, call the methods SPEED_UP and WRITE once for each entry in the internal table. When you call SPEED_UP, pass the value SY-TABIX * 10 to the parameter STEP.

Exercise 3 - Inheritance

- Change your program from exercise 2 to define classes TRUCK and SHIP as direct subclasses of VEHICLE.
- The class TRUCK should have an instance constructor and redefine the method WRITE.
- The class SHIP should have an IMPORTING parameter NAME, a new public attribute NAME, and should also redefine the method WRITE.
- The instance constructor of each class should change the maximum speed. The instance constructor of SHIP should set the attribute NAME to the actual parameter that you imported.
- The WRITE method should show the class from which the display comes. For SHIP, use the NAME attribute.
- Declare extra reference variables TRUCK and SHIP for the new classes.
- You can delete the code that creates objects for VEHICLE. Instead, create one instance of each of your new classes and place the corresponding reference into VEHICLE_TAB.
- Call the method SPEED_UP for both classes using the correct subclass reference, and WRITE using a superclass reference.

Exercise 4 - Interfaces

- Change your program from exercise 3 so that the method WRITE is contained in an interface STATUS.
- The class VEHICLE should implement the interface instead of its own WRITE method. The name must be changed accordingly in each implementation (including the subclasses), but the function remains the same.
- Replace the reference variables VEHICLE and the table VEHICLE_TAB with the interface reference STATUS and internal table STATUS_TAB.
- Use these references to display the status of the objects.
- Create a new class HELICOPTER that also implements STATUS. Declare an alias WRITE in that class for the interface method.
- The interface method in HELICOPTER should display a different line from that displayed in VEHICLE.
- Declare a reference variable HELI for the class HELICOPTER, create a corresponding object, and insert the reference variable into the table STATUS_TAB.

Exercise 5 - Events

- Change your program from exercise 4 to include an event DAMAGED for the class SHIP.
- Redefine the method SPEED_UP in the class SHIP.
- When the maximum speed is exceeded, SPEED_UP should set the maximum speed to zero and trigger the event DAMAGED.
- Add a new method RECEIVE to the class HELICOPTER to allow it to handle the event DAMAGED. Import the implicit parameter SENDER.
- RECEIVE should use the SENDER parameter to display the name of the ship that triggered the event.
- Register the handler method of the object to which HELI is pointing as a handler for objects from the class SHIP.
- Increase the speed of the object from the class SHIP until it exceeds the maximum speed and triggers the event.