

ABAP Part II

Lesson 3: Object oriented ABAP

Lesson Objectives



After completing this lesson, participants will be able to understand -

- OOPS Concepts
- ABAP Objects
- Creating & Accessing objects
- Constructor
- Inheritance
- Casting
- Interfaces
- Events
- Exceptions





Object-oriented programming, is a problem-solving method in which the software solution reflects objects in the real world.

Benefits of Object-oriented programming are :

- Multiple Instances
- Encapsulation
- Inheritance
- Polymorphism
- Compatibility
- Maintainability



Benefits of Object-oriented programming

Multiple Instances

- The ability to create multiple instances of a "class", such as a vehicle, is one of the central attributes of object-oriented languages.

Encapsulation

- Encapsulation means that the implementation of an object is hidden from other components in the system, so that they cannot make assumptions about the internal status of the object and therefore dependencies on specific implementations do not arise



Polymorphism

- Polymorphism (ability to have multiple forms) in the context of object technology signifies that objects in different classes react differently to the same messages.

Inheritance

- Inheritance defines the implementation relationship between classes, in which one class (the subclass) shares the structure and the behavior defined in one or more other classes (super classes).
- Note: ABAP Objects only allows single inheritance.

Compatibility

- ABAP object is true extension of ABAP language. ABAP OOPS statements can be used in procedural ABAP programs. Object themselves can contain classic ABAP statements, Only OOPS concepts that have been proved useful have been included. It has been kept the simplest. There is increased use of type checks.

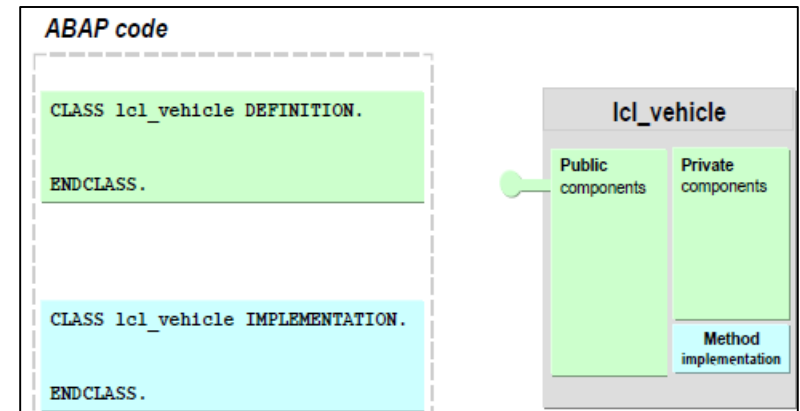
ABAP Objects - Class



A class is a set of objects that have the same structure and the same behavior. A class is therefore like a blueprint, in accordance with which all objects in that class are created.

The components of the class are defined in the definition part. The components are attributes, methods, events, constants, types, and implemented interfaces. Only methods are implemented in the implementation part.

The CLASS statement cannot be nested, that is, you cannot define a class within a class.





Classes are the central element of object-orientation.

A Class is an abstract description of an object.

Classes are templates for objects.

Defines the state and behavior of the object.

Types of classes

- Local classes
 - Defined within an ABAP program
 - Can be used only within that program
- Global classes
 - Defined in the class builder SE24
 - Stored centrally in class library in the R/3 repository
 - Can be accessed from all the programs in the R/3 system
 - e.g. CL_GUI_ALV_GRID, CL_GUI_CUSTOM_CONTAINER



Attributes describe the data that can be stored in the objects of a class.

Attributes can have any kind of data type:

- C, N, I, P, ..., STRING
- Dictionary types
- User-defined types
- TYPE REF TO defines a reference to an object, in this case "r_car"

Public attributes

- Can be viewed and changed by all users and in all methods
- Direct access

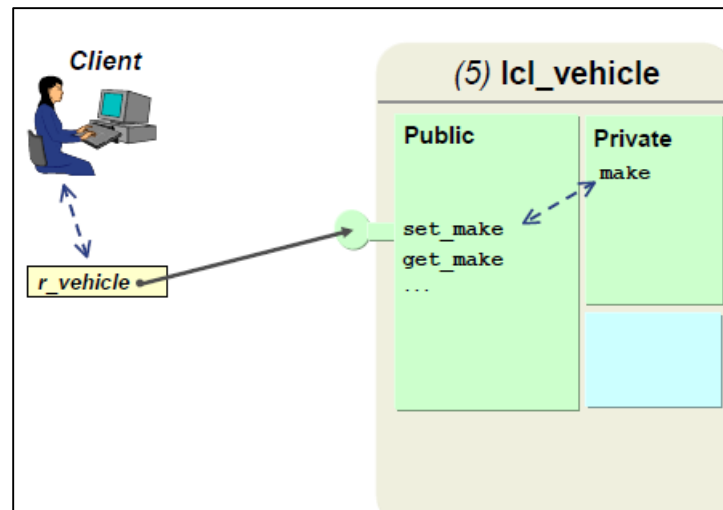
Private attributes


- Can only be viewed and changed from within the class
- No direct access from outside the class

ABAP Objects - Attributes




Accessing private attributes :You can access an object's private attributes using public methods, which in turn output this attribute or change it.



```
CLASS lcl_vehicle DEFINITION.  
  PUBLIC SECTION.  
    DATA: make TYPE string.  
  PRIVATE SECTION.  
ENDCLASS. 
```

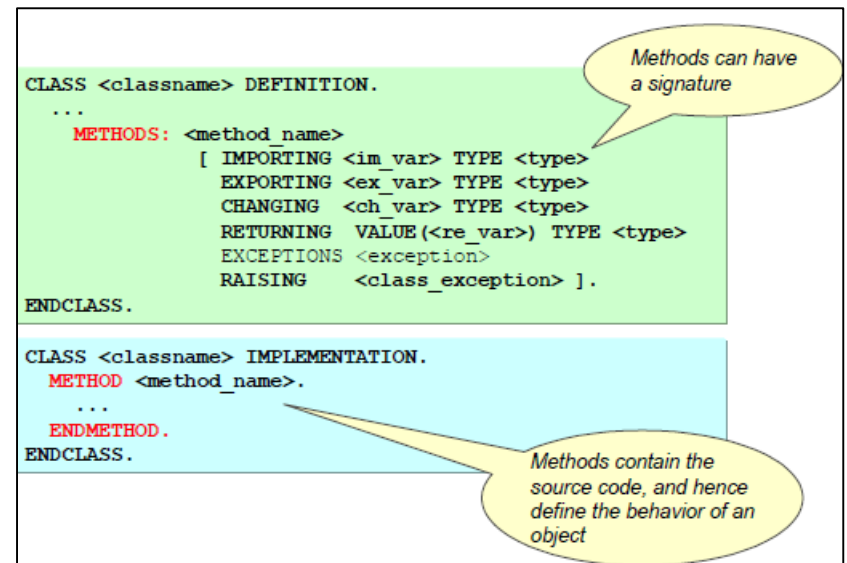


```
CLASS lcl_vehicle DEFINITION.  
  PUBLIC SECTION.  
    ...  
  PRIVATE SECTION.  
    DATA: make TYPE string.  
ENDCLASS. 
```



ABAP Objects - Methods

1. Methods are internal procedures in classes that determine the behavior of an object. They can access all attributes in their class and can therefore change the state of an object.
2. Methods have a parameter interface (called signature) that enables them to receive values when they are called and pass values back to the calling program.





Methods and Visibility

Public methods

- Can be called from anywhere

Private methods

- Can only be called within the class

Demo: Create a class with instance attributes and instance methods





Instance attributes and Static attributes

Instance attributes

- One per instance
- Statement: DATA

Static attributes

- Only one per class
- Statement: CLASS-DATA
- Also known as class attributes

```
CLASS lcl_vehicle DEFINITION.  
  
    PUBLIC SECTION.  
        ...  
    PRIVATE SECTION.  
        DATA: make      TYPE string,  
        ...  
  
        CLASS-DATA: n_o_vehicles TYPE i.  
  
ENDCLASS.
```



Instance method and Static method

Instance methods

- Can use both static and instance components in their implementation part
- Can be called using an instance

Static methods

- Can only use static components in their implementation part
- Can be called using the class

Demo: Create a class with static attributes and static method

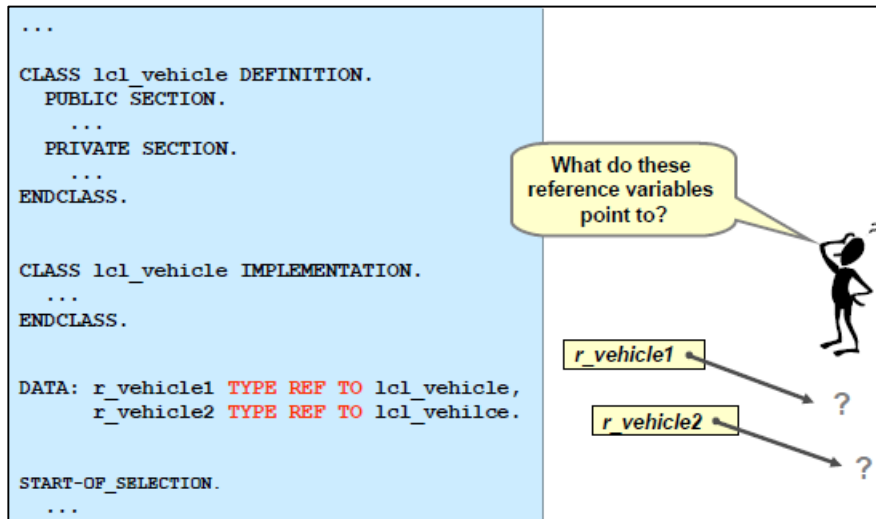


Reference Variable



A reference variable acts as a pointer to an object.

- DATA: R_VEHICLE1 TYPE REF TO LCL_VEHICLE.
- Declares a reference variable that acts as a pointer to an object

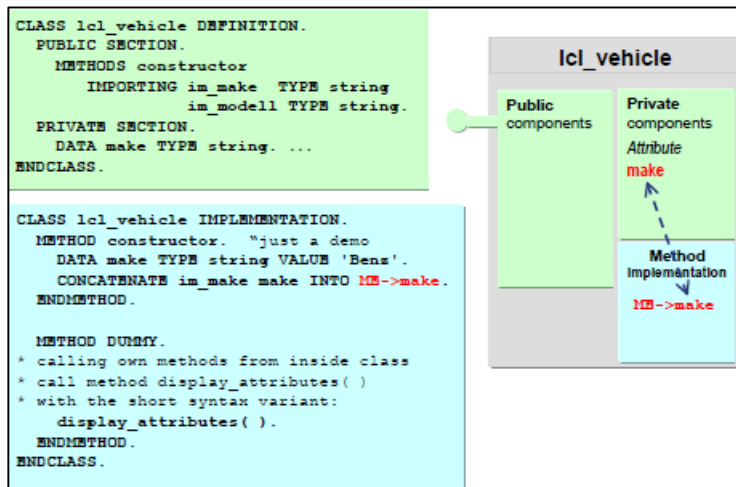


Reference Variable ME



You can address the object itself within instance methods using the implicitly available reference variable `me`.

Description of example: In the constructor, the instance attribute `make` is covered by the locally defined variable `make`. In order to still be able to address the instance attribute, you need to use `me`.



Demo: Create a class with Reference variable Me



Creating and Accessing Objects

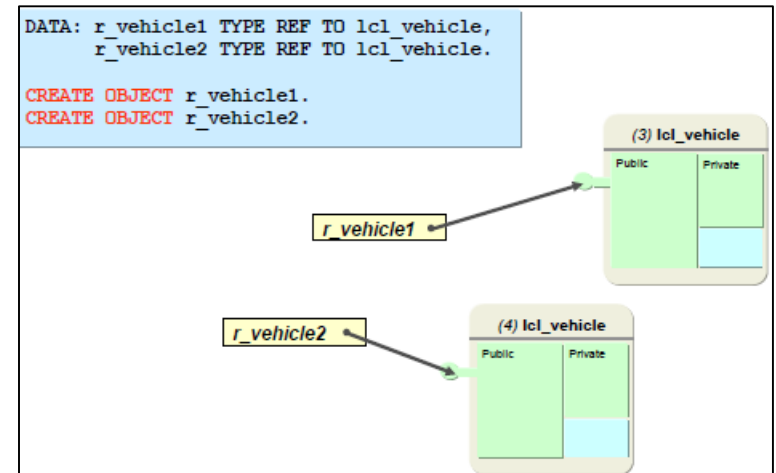
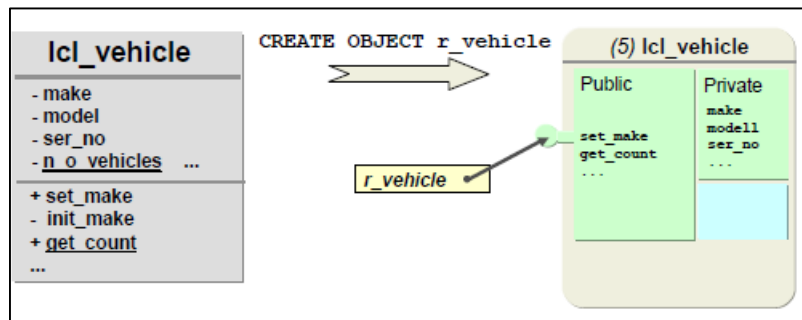


An object is a section of source code that contains data and provides services

Objects are created using the CREATE OBJECT statement

Objects can only be created and addressed using reference variables

An object is a section of source code that contains data and provides services.



Accessing Attributes and Method



Instance methods are called using

CALL METHOD <reference>-><instance_method>.

Static methods (also referred to as class methods) are called using

CALL METHOD <classname>=><class_method>.

If you are calling a static method from within the class, you can omit the class name.

Static attributes are accessed using

<classname>=><class_attribute>

instance attributes are accessed using

<instance>-><instance_attribute>

=> and -> are the component selectors



Methods that have a RETURNING parameter are described as functional methods. These methods cannot have EXPORTING or CHANGING parameters, but has many (or as few) IMPORTING parameters and exceptions as required. You can only do this for a single parameter, which additionally must be passed as a value.

```
CLASS lcl_vehicle DEFINITION.  
  PUBLIC SECTION.  
    METHODS: get_average_fuel  
              IMPORTING im_distance    TYPE s_distance,  
                      im_fuel         TYPE ty_fuel  
              RETURNING VALUE(re_fuel) TYPE ty_fuel,  
ENDCLASS.  
  
DATA: r_vehicle1 TYPE REF TO lcl_vehicle,  
      r_vehicle2 TYPE REF TO lcl_vehicle,  
      avg_fuel TYPE ty_fuel.  
  
...  
* example for short syntax in aritmet. operation  
avg_fuel =  
  r_vehicle1->get_average_fuel( im_distance = 500 im_fuel = 50 )  
+ r_vehicle2->get_average_fuel( im_distance = 600 im_fuel = 60 ).
```

Demo: Create a class with Functional Method





The constructor is a special instance method in a class with the name constructor.

Each class can have one constructor.

The constructor is automatically called at runtime within the CREATE OBJECT statement.

If you need to implement the constructor, then you must define and implement it in the PUBLIC SECTION.

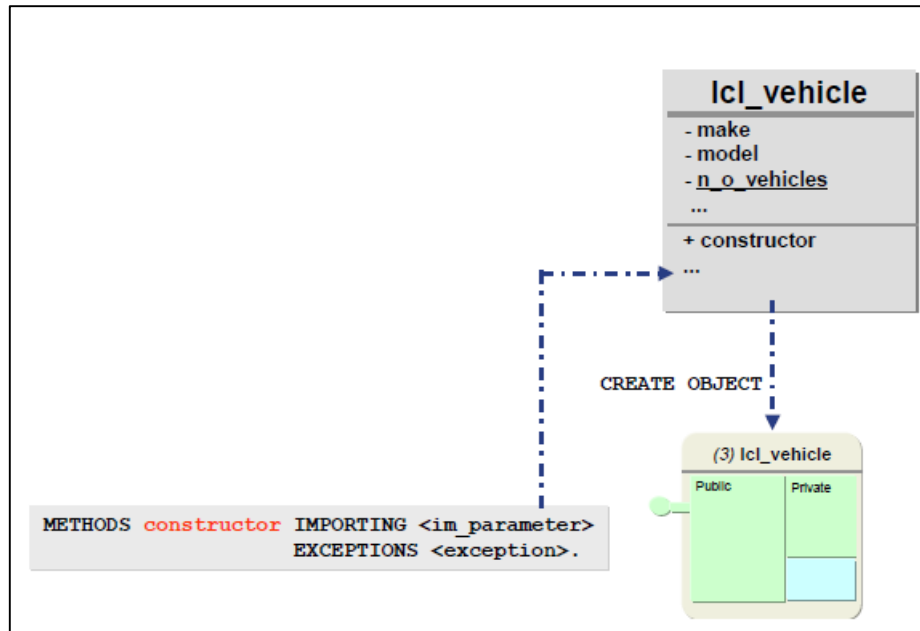
You cannot normally call the constructor explicitly

Special method for creating objects with defined initial state

Only has IMPORTING parameters and EXCEPTIONS

Is executed only once per instance

Constructor



Demo: Create a class with Constructor





The static constructor is a special static method in a class with the name `class_constructor`. It is executed precisely once per program.

The static constructor of a class `<classname>` is called automatically when the class is first accessed, but before any of the following actions are executed:

Creating an instance in the class using `CREATE OBJECT <obj>`, where `<obj>` has the data type `REF TO <classname>`

Addressing a static attribute using `<classname>=><attribute>`

Calling a static attribute using `CALL METHOD <classname>=><classmethod>`

Registering a static event handler method using `SET HANDLER <classname>=><handler_method> FOR <obj>`

Registering an event handler method for a static event in class `<classname>`.

The static constructor cannot be called explicitly.

Demo: Create a class with Static Constructor



Demo: Global Class creation through SE24



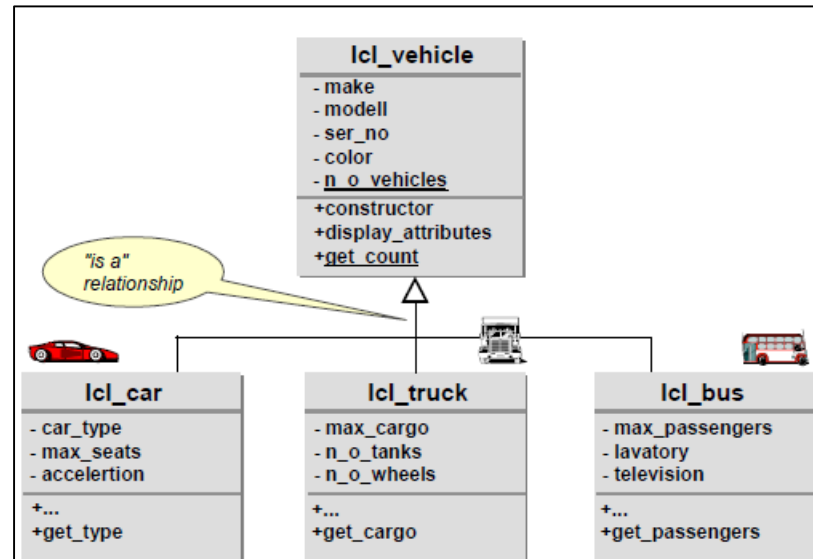
Inheritance



Inheritance is a relationship, in which one class (the subclass) inherits all the main characteristics of another class (the superclass). The subclass can also add new components (attributes, methods, and so on) and replace inherited methods with its own implementations.

The inheritance relationship is often described as an "is a" relationship:

- A truck is a vehicle





Single Inheritance

- ABAP Objects only has single inheritance.
- A class may only have one direct superclass, but it can have more than one direct subclass. The empty class OBJECT is the root node of every inheritance tree in ABAP Objects.



Relationship between Super class and Sub class

- Common components only exist once in the superclass
 - New components in the superclass are automatically available in subclasses
 - Amount of new coding is reduced ("programming by difference")
- Subclasses are extremely dependent on superclasses
- "White Box Reuse":
 - Subclass must possess detailed knowledge of the implementation of the superclass



Normally the only other entry required for subclasses is what has changed in relation to the direct superclass. Only additions are permitted in ABAP Objects, that is, in a subclass you can "never take something away from a superclass". All components from the superclass are automatically present in the subclass.



The REDEFINITION statement for the inherited method must be in the same SECTION as the definition of the original method.

If you redefine a method, you do not need to enter its interface again in the subclass, but only the name of the method.

In the case of redefined methods, changing the interface (overloading) is not permitted; exception: Overloading is possible with the constructor.

Within the redefined method, you can access components of the direct superclass using the SUPER reference.

The pseudo-reference super can only be used in redefined methods.

Demo: Inheritance





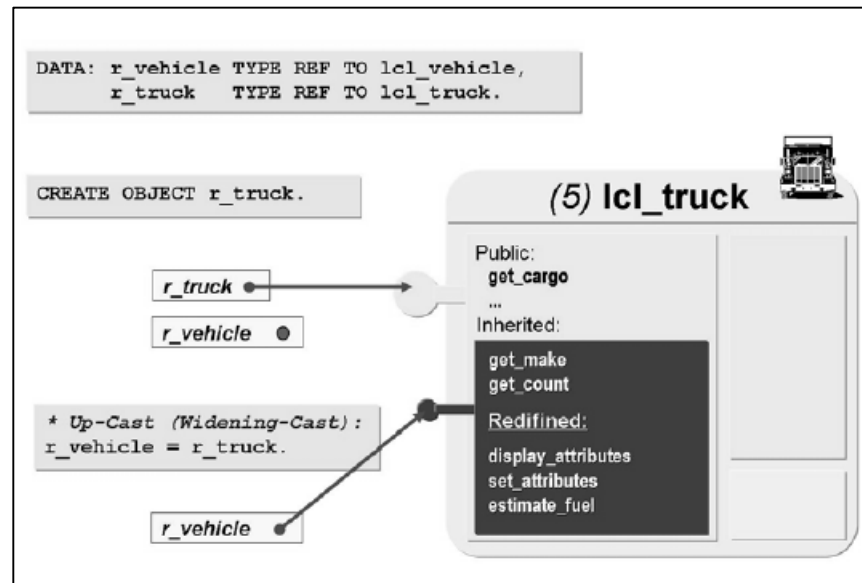
Up-Cast (Widening Cast)

- The assignment of a subclass instance to a reference variable of the type "reference to superclass" is described as a Widening cast. As the target variable can accept more dynamic types in comparison to the source variable, this assignment is also called widening cast.

What is a Widening cast used for?

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

Widening Cast



Demo: Inheritance – Widening cast





Down-cast (Narrowing Cast)

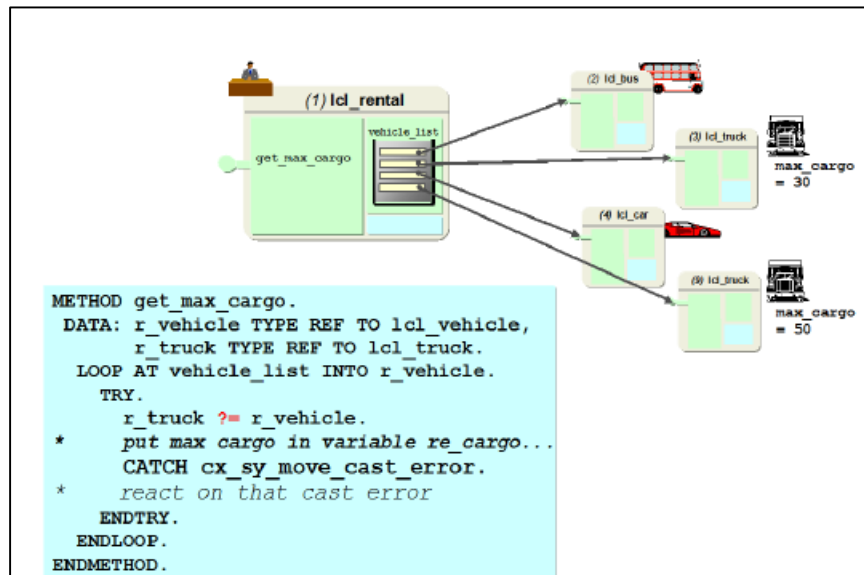
- The Narrowing cast logically represents the opposite of the widening cast. The narrowing cast cannot be checked statically, only at runtime. The Cast Operator `?=` (or the equivalent `MOVE ... ?TO...`) must be used to make this visible.
- As the target variable can accept less dynamic types after the assignment, this assignment is also called narrowing cast.

What is a Narrowing cast used for?

- The client, the car rental company wants to execute a function for specific vehicles from the list (`vehicle_list`). For example, the client wants to ascertain the truck with the largest cargo capacity. However, not all vehicles are in the trucks list, it also includes references to cars and busses.



With this kind of cast, a check is carried out at runtime to ensure that the current content of the source variable corresponds to the type requirements of the target variables. If it is, the assignment is carried out. Otherwise, an exception of the error class `CX_SY_MOVE_CAST_ERROR` is raised.



Demo: Inheritance – Narrowing cast



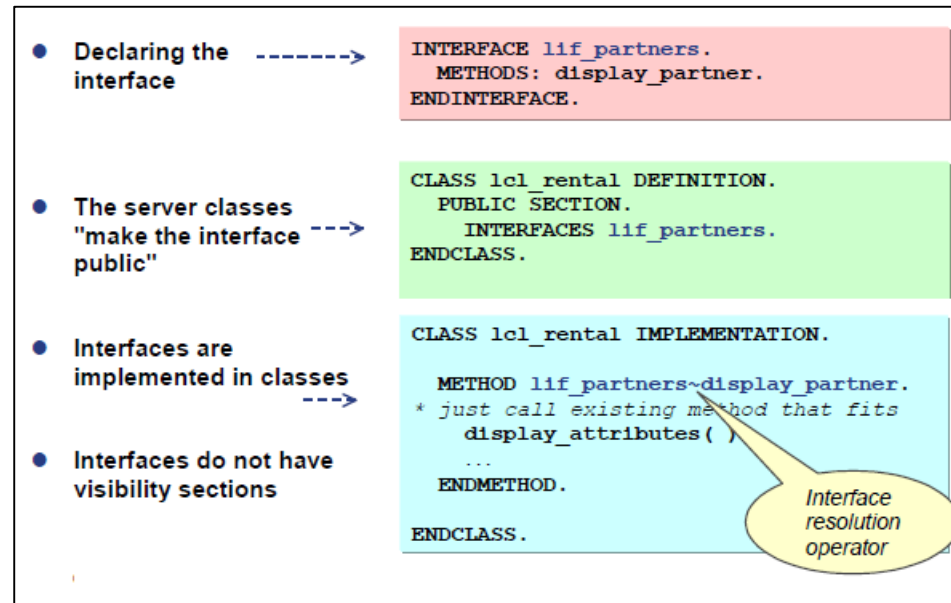


Interfaces only describe the external point of contact of a class (protocols), they do not contain any implementation.

Interfaces are usually defined by a user. The user describes in the interface which services (technical and semantic) it needs in order to carry out a task.

The user never actually knows the providers of these services, but communicates with them through the interface.

In this way the user is protected from actual implementations and can work in the same way with different classes/objects, as long as they provide the services required. This is known as polymorphism with interfaces.



Demo: Interfaces





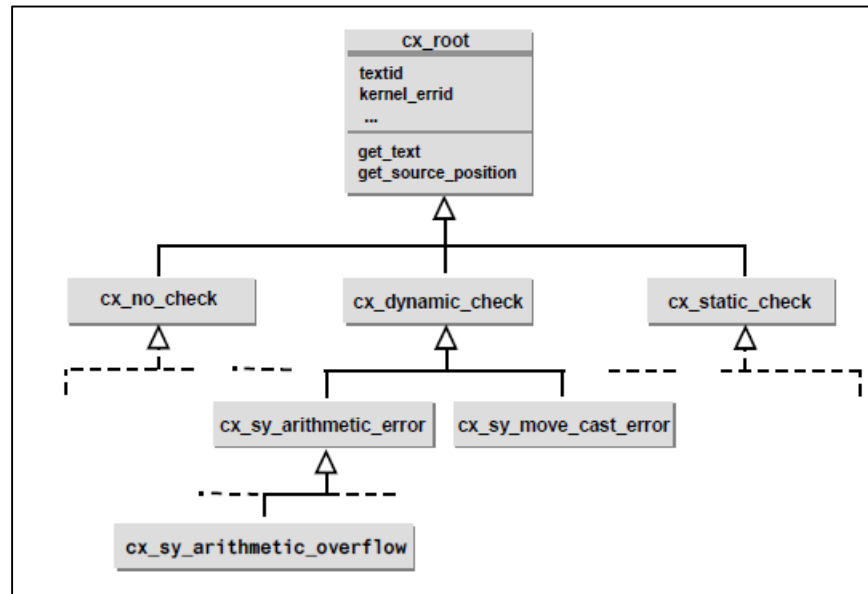
Exception refer to a situation that arises while an ABAP program is being executed, where there is no point in continuing to run the program in the normal way.

Class-based exceptions are raised either using the ABAP statement `RAISE EXCEPTION` or by the ABAP runtime environment.

If a class-based exception occurs, the system interrupts the normal program flow and tries to navigate to a suitable handler. If it cannot find a handler, a runtime error occurs.

The use of class-based exceptions is not limited to object-oriented contexts. Class-based exceptions can be raised and handled in all ABAP processing blocks.

Exceptions Classes: Inheritance hierarchy



Exceptions Classes: Inheritance hierarchy



The root class `CX_ROOT` contains two predefined methods that are inherited by the other classes.

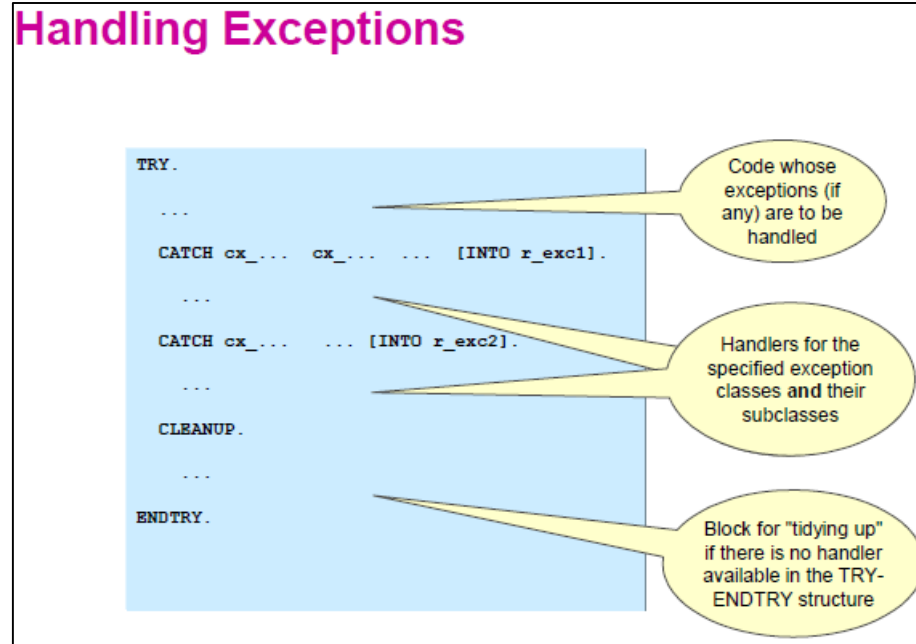
The `GET_SOURCE_POSITION` method returns the program name, include name (if relevant), and line number in the source code where the exception occurred.

The `GET_TEXT` method returns an exception text of a class in the form of a string.

You can assign several texts to each class. You can then specify which text is to be used when an exception is raised by passing an identifier to the `IMPORTING` parameter `TEXTID` of the instance constructor.

All exception classes inherit the `KERNEL_ERRID` attribute from `CX_ROOT`. This attribute contains the name of the appropriate runtime error if the exception was raised by the runtime environment - such as `COMPUTE_INT_ZERODIVIDE` if the program tries to divide by zero.

Handling Exceptions



Handling Exceptions



The TRY block contains the application code that is to handle the exceptions.

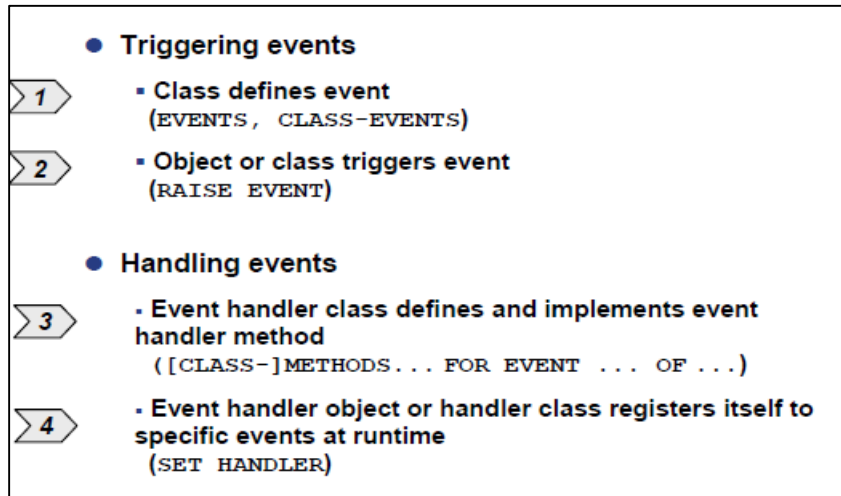
If an exception occurs in the TRY block the system searches first for a CATCH statement (which will handle the exception) in the same TRY-ENDTRY structure and then step by step outwards in all the enclosing TRY-ENDTRY structures.

A CATCH block contains the exception handler that is executed if a specified exception has occurred in the TRY block in the same TRY-ENDTRY structure.

In some cases, the system cannot find a handler for an exception within a specific TRY-ENDTRY structure but the exception is handled in a surrounding TRY-ENDTRY structure or passed along to a calling program. If this occurs, a CLEANUP block is executed before leaving the TRY-ENDTRY structure.



By triggering an event, an object or class announces a change of state, or that a certain state has been achieved.





At the moment of implementation, a class defines its :Instance events (using the EVENTS statement) Static events (using the CLASS-EVENTS statement)

Classes or their instances that receive a message when an event is triggered at runtime and want to react to this event define event handler methods. Statement: [CLASS-]METHODS <handler_method> FOR EVENT <event> OF <classname>.

These classes or their instances are registered to one or more events at runtime. Statement: SET HANDLER <handler_method> FOR <reference>. (for instance events) SET HANDLER <handler_method>. (for static events)

A class or instance can trigger an event at runtime using the RAISE EVENT statement.



Both instance and static events can be triggered in instance methods. Only static events can be triggered in static methods.

Events can only have EXPORTING parameters which must be passed by value.

Triggering an event using the statement RAISE EVENT has the following effect:

- 1.The program flow is interrupted at that point
- 2.The event handler methods registered to this event are called and processed
- 3.Once all event handler methods have been executed, the program flow continues



Registering for an event

- When an event is triggered, only those event handler methods are executed that have, by this point, registered themselves using SET HANDLER.
- You can register an event using ACTIVATION 'X', and deregister it using ACTIVATION space. If you do not specify ACTIVATION, then the event registers (default behavior).
- You can register several methods in one SET HANDLER statement: SET HANDLER <ref_handle1>-><handler_method1> ... <ref_handleN>-><handler_methodN> FOR <ref_sender> | FOR ALL INSTANCES



Registration/De registration: Handler Tables

- Every object that has defined events has an internal table, the handler table. All objects that have registered for events are entered in this table together with their event handler methods.

Demo: Events



Summary



In this lesson, you have learnt:

- OOPS Concepts
- ABAP Objects
- Creating & Accessing objects
- Constructor
- Inheritance
- Casting
- Interfaces
- Events
- Exceptions

Review Question



Question 1: _____ do not contain implementation.

Question 2: _____ means that the implementation of an object is hidden from other components in the system.