Object Oriented Programming in SAP-ABAP

ABAP Objects:

Overview

Definition

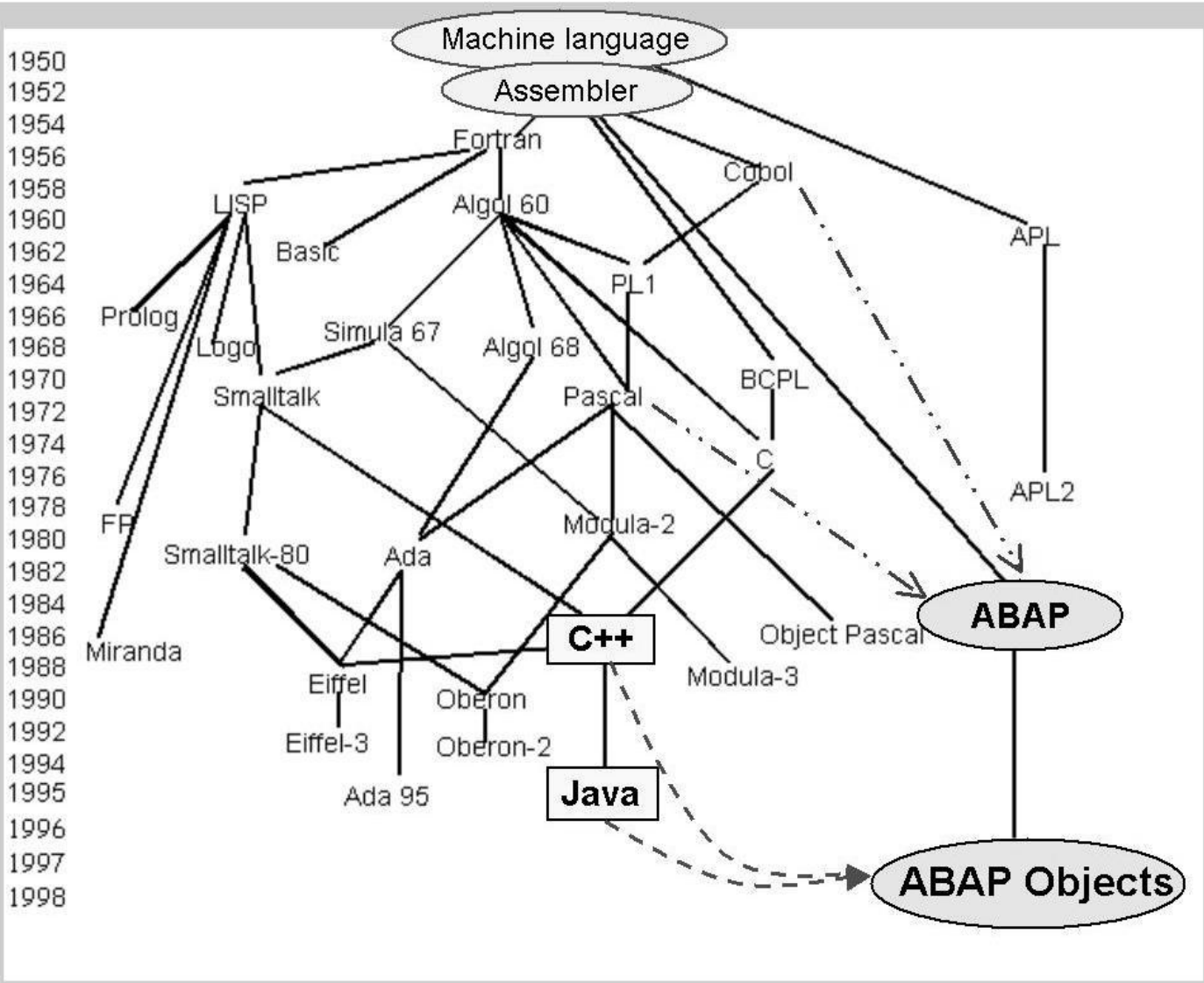
ABAP

- Advanced Business Application Programming
- Is a historically grown 4GL language
- Is specialized for business applications

ABAP Objects

- Object-oriented extension of ABAP programming language

History


1950
1952
1954
1956
1958
1960
1962
1964
1966
1968
1970
1972
1974
1976
1978
1980
1982
1984
1986
1988
1990
1992
1994
1995
1996
1997
1998
Machine language
Assembler
Fortran
Cobol
LISP
Algol 60
APL
Basic
PL1
Prolog
Logo
Simula 67
Algol 68
BCPL
Smalltalk
Pascal
C
APL2
FP
Modula-2
Smalltalk-80
Ada
ABAP
C++
Object Pascal
Miranda
Modula-3
Eiffel
Oberon
Eiffel-3
Oberon-2
Java
Ada 95
ABAP Objects

ABAP Programming Model

ABAP supports a hybrid programming model:

- A object-oriented programming model that is based on classes and interface of ABAP objects
- A procedural programming model that is based on procedural calls and system event handling
- Both models are interoperable – you can call classes from classic procedures, and you can call classic procedures from methods.

ABAP Objects:

Concept

Object-oriented Programming

Abstraction

- Ability to reflect real-world processes

- real-world modeled in classes and mapped in objects

Encapsulation

- Implementation details are hidden behind interfaces.

- Ensure that the abstract representation of an object is used only in accordance with its specification.

Inheritance

- Abstractions derived from existing ones.

Instantiation

- Enables you to create multiple instances of a class.

Why use ABAP Objects?

- ABAP Objects results in a clean and secure programming style.
- SAP delivers essential APIs (ex. Control Framework) in object-oriented wrapped classes.
- Most SAP new products are written in ABAP Objects!

Object-oriented Programming Basics

- Class is the model or the template
- Object is an *instance of class*
- **Local class is created within any ABAP program and only visible there**
- **Global class is created with the Class Builder tool and visible in any ABAP program**

ABAP Objects:

Keywords and Syntax

Define a CLASS

CLASS class_name DEFINITION.

....

ENDCLASS.

CLASS class_name IMPLEMENTATION.

....

ENDCLASS.

Visibility Section

Public

- All components are public and can be addressed by all subclasses, the class itself and other external classes.

Protected

- Components of this section are protected and can only be addressed by the class itself and the subclasses

Private

- Can only be used in the methods of the class itself

Define Visibility Section


```
CLASS class_name DEFINITION.
    PUBLIC SECTION.
        ....
    PROTECTED SECTION.
        ....
    PRIVATE SECTION.
        ....
ENDCLASS.
```

Controlling the Instantiation

Public instantiation

- Allow to any user to create objects (default)

Protected instantiation

- Allow the creation of objects in methods of subclasses

Private instantiation

- Only the class itself can create objects

Define Instantiation

```
CLASS class_name
        DEFINITION CREATE
        PUBLIC | PROTECTED | PRIVATE.

ENDCLASS.
```

Components

Components of a class

1. Attributes
2. Methods
3. Events

Components – Attribute

- The data objects within a class

- All data types of the ABAP type hierarchy can be used

- ‘DATA’ statement is used for instance attributes

- ‘CLASS-DATA’ for static attribute

Define Attributes

```
CLASS attributes DEFINITION.

    PRIVATE SECTION.

    DATA objectValue TYPE i.

    CLASS-DATA objectCount TYPE i.

ENDCLASS.
```

Components – Methods

- Processing block with a parameter interface

- Local data

- Instance method can access all attributes and events

- Static method can access only static attributes and events

Define Instance Methods

```
CLASS class_name DEFINITION.

  PUBIC SECTION.

    METHODS meth

      IMPORTING    .. Ii  TYPE  type

      EXPORTING    .. Ei TYPE type ..

      CHANGING     .. Ci TYPE type ..

      EXCETPIONS .. Ei

ENDCLASS.
```

Define Static Methods

```
CLASS class_name DEFINITION.

  PUBIC SECTION.

    CLASS-METHODS class_meth

        IMPORTING    .. Ii  TYPE  type

        EXPORTING    .. Ei TYPE type ..

        CHANGING     .. Ci TYPE type ..

        EXCETPIONS .. Ei

ENDCLASS.
```

Implementation of Methods

```
CLASS class_name DEFINITION.
    METHOD meth.
        ...
    ENDMETHOD.
ENDECLASS.
```

Components – Methods

Two special methods

constructor

- implicitly called for each instantiation of a new object
- Execute by runtime environment.

class_constructor

– first time a class is accessed

Destructors

ABAP Objects *does not* provide any destructors for application development.

Define Object Reference Variables

To create and access an object, object references in reference variables are needed.

Syntax:

DATA ref TYPE REF TO class

Instantiating

Syntax:

CREATE OBJECT ref.

Different Selectors

Accessing fields of a structure

- Structure component selector “**-**”

Accessing object components

- Object component selector “**->**”
- Class component selector “**=>**”
- Interface component selector “**~**”

Use of Methods

Call instance method:

Syntax:

```
CALL METHOD oref->get_data.
```

Call class method:

Syntax:

```
CALL METHOD class=>get_data.
```

Functional method

- Method with any number of input parameters but only one return value.
- It can be used in operand positions for functions and expressions.

Use of Functional Method

Functional method used in expressions:

Syntax

```
IF obj->functional_method( ) = abap_true.
        ……. do some thing
ENDIF.
```

Method Call

```
CALL METHOD oref->get_data.
CALL METHOD class=>get_data.

CALL METHOD oref->get_data
      IMPORTING P1 = P1
      RECEIVING P2 = P2.
```

Method Call Short Form
- Functional Method Call Style

```
oref->get_data( ).
class=>get_data( ).

P2 = oref->get_data( P1 ).
```

Special Reference Variables

The self-reference:

- me

The superclass-reference(used in method call):

- super

CLASS DEFINITION DEFERRED

CLASS - DEFERRED, LOAD

Syntax

CLASS class DEFINITION

DEFERRED | *LOAD* .

CLASS DEFINITION DEFERRED

```
CLASS c1 DEFINITION DEFERRED.

CLASS c2 DEFINITION.
  PUBLIC SECTION.
    DATA c1ref TYPE REF TO c1.
ENDCLASS.

CLASS c1 DEFINITION.
  PUBLIC SECTION.
    DATA c2ref TYPE REF TO c2.
ENDCLASS.

CLASS cl_gui_cfw DEFINITION LOAD.

DATA state LIKE cl_gui_cfw=>system_state.
```

Components – Events

- Allow the objects of a class to publish its status. Other objects can then respond to the change in status.

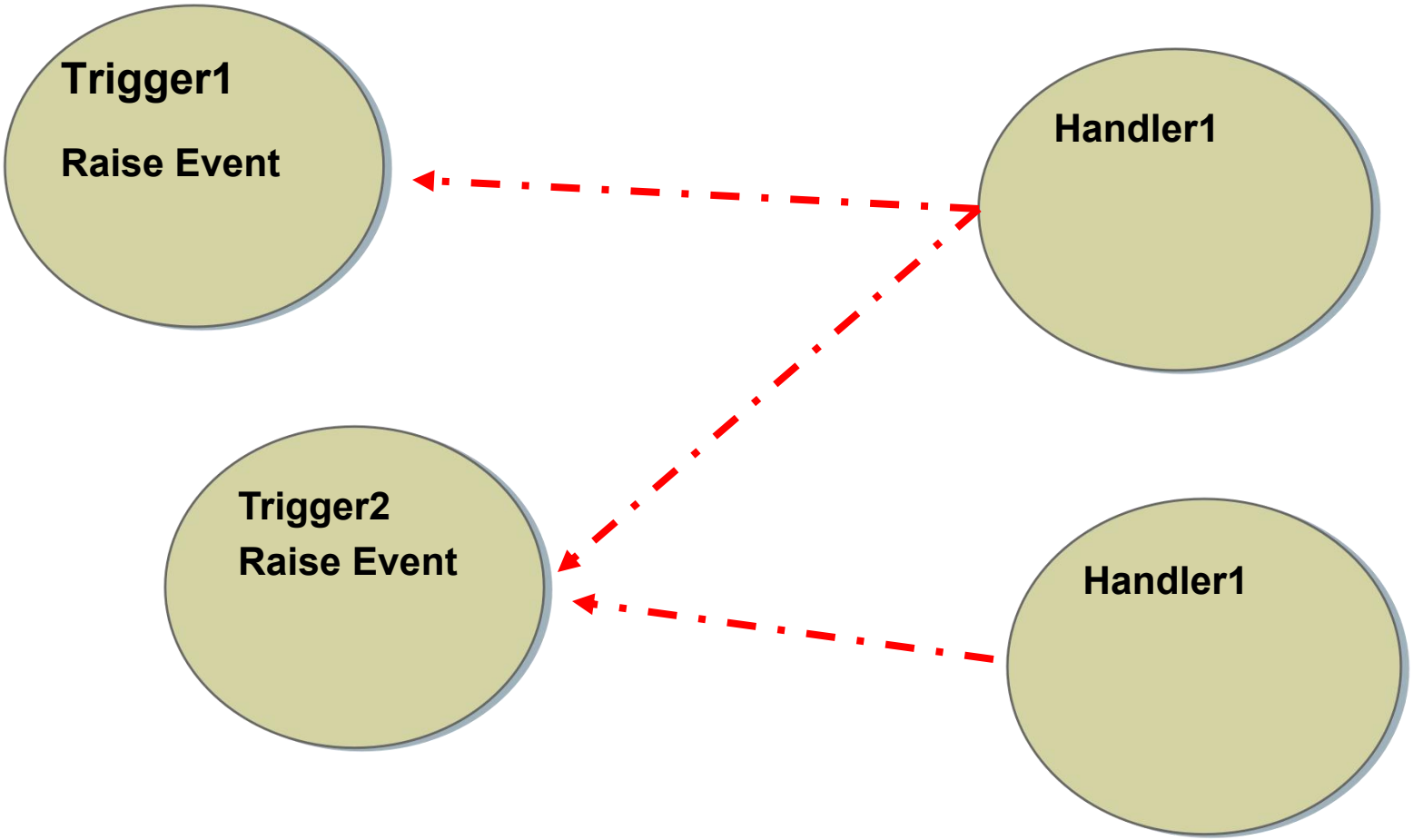
1. Event trigger does not know the event handler.

2. Event trigger therefore does not initially

know if the event will even have an effect.

- Publish and Subscribe mechanism, i.e. Broadcasting

Event Triggers and Event Handlers



Terms

Events

Each method can trigger the events of its class

Handler methods

Methods can be declared as handler methods for events

Registering handlers

Objects for an event can be registered at runtime as handlers for the objects of the classes which can trigger the event

Define Events

```
CLASS raising_class DEFINITION.
    PUBLIC SECTION.
        EVENTS: raised_event.
ENDCLASS.
```

Raise Events

```
CLASS raising_class IMPLEMENTATION.
    METHOD raising_method.
        RAISE EVENT raised_event.
    ENDMETHOD.
ENDCLASS.
```

Define Handler Methods

```
CLASS handler_class DEFINITION.
    PUBLIC SECTION.
        METHODS: handler_method FOR EVENT
            raised_event OF raising_class.
ENDCLASS.
```

Handler Methods

```
CLASS handler_class IMPLEMENTATION.
  METHOD handler_method.
    … some logics here
  ENDMETHOD.
ENDCLASS.
```

Register Handlers

```
SET HANDLER
    handler_instance->handler_method
        FOR ALL INSTANCES.

SET HANDLER
    handler_instance->handler_method
        FOR handled_instance.
```

ABAP Objects:

Exception Handling

Exception Handling

Exceptions are events that arise during the execution of an ABAP program at which the program is interrupted, because it is not possible to continue processing the program in a meaningful way.

Exceptions are either treatable or untreatable.

Exception Handling

Runtime Errors

Exceptions Before Release 6.10

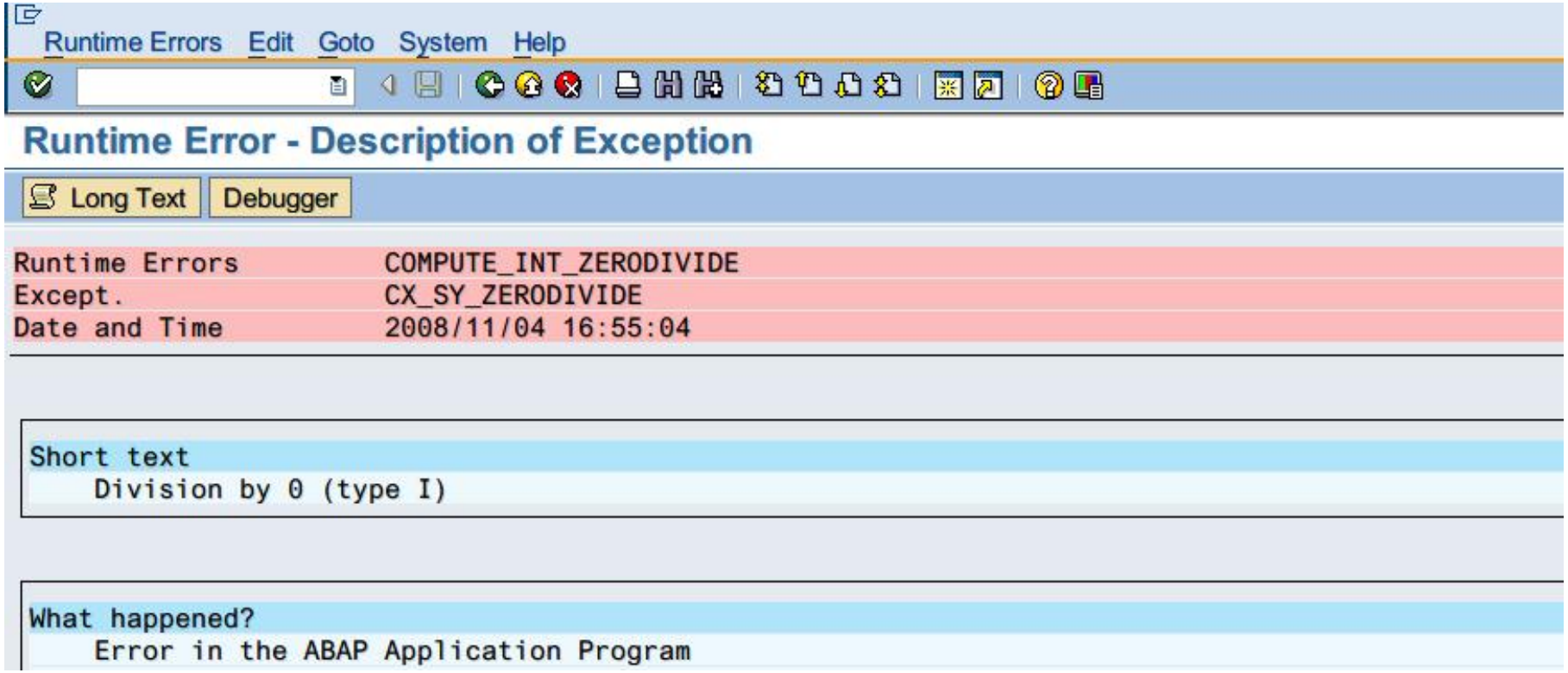
Class-Based Exceptions

Runtime Errors

Exception that was not handled

```
data RESULT type I.
RESULT = 1 / 0.
```

Shortdump screenshot



Exceptions Before Release 6.10

Handling exceptions as catchable runtime errors

```
data RESULT type I.
catch system-exceptions ARITHMETIC_ERRORS = 4.
  RESULT = 1 / 0.
endcatch.
if SY-SUBRC = 4.
  ...
endif.
```

Class-Based Exceptions

Handling exceptions using/with exception classes

```
data MYREF type ref to CX_SY_ARITHMETIC_ERROR.
data ERR_TEXT type STRING.
data RESULT type I.
try.
    RESULT = 1 / 0.
  catch cx_sy_arithmetic_error into MYREF.
    ERR_TEXT = MYREF->GET_TEXT( ).
endtry.
```

Defining Exceptions

All exception classes must inherit from the common superclass CX_ROOT and one of its subordinate classes:

- CX_STATIC_CHECK
- CX_DYNAMIC_CHECK
- CX_NO_CHECK

Propagating Exceptions

The **RAISING** addition is specified in the declaration of a method / subroutine:

METHODS meth ... RAISING cx_... cx_...

FORM form ... RAISING cx_... cx_...

Use of ABAP Objects:

Advanced Concept

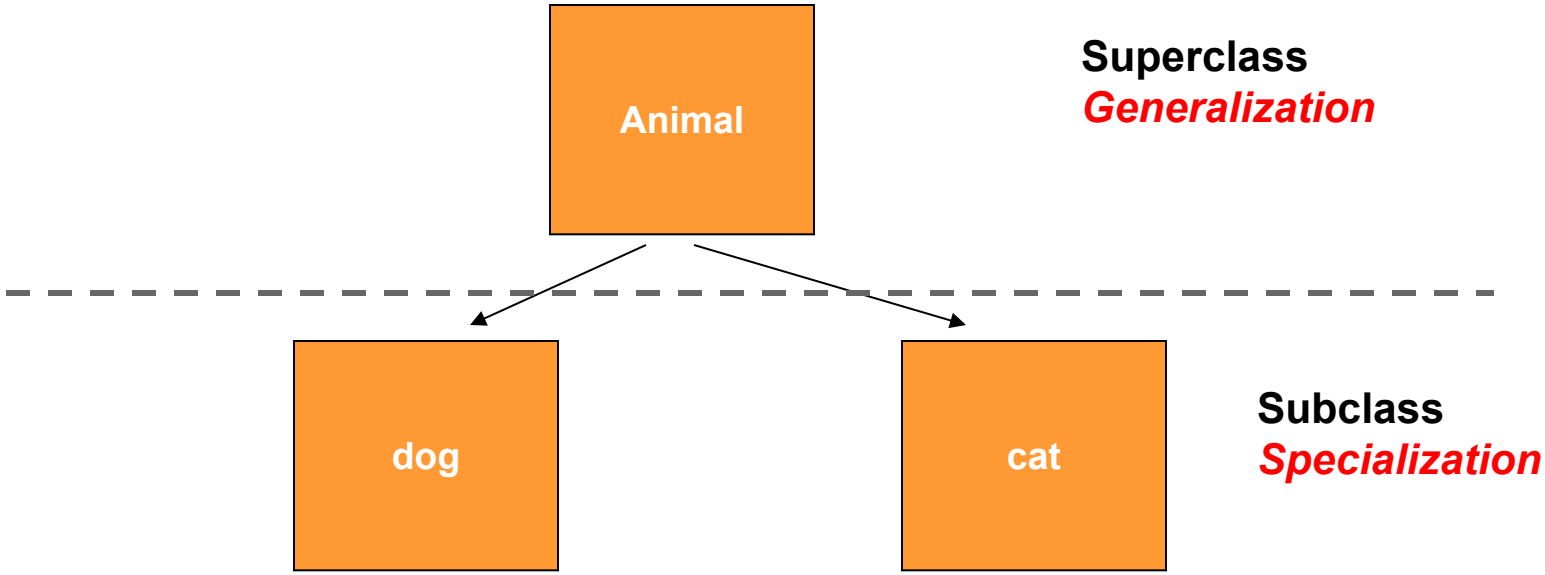
Advanced Concept of Object-oriented Programming

Inheritance

Interface

Friend

Inheritance



Visibility in Inheritance

Public - contains all the public components of all superclasses and its own added public components.

Protected - Contains all the protected components of all superclasses as well as its own.

Private - Contains its own private components. Address by its own methods.

Method Redefinition

Re-implement the method of supper class

In Class declaration part:

Syntax:

METHODS meth REDEFINITION.

Up Cast

Up Cast (Also referred to as widening cast)

- *Assignment between reference variables, where the static type of the target variables is similar or identical to the static type of the source variables.*

Casting operator

= or MOVE ... TO … .

Down Cast

Down Cast (Also called a narrowing cast)

- an assignment between reference variables in which the static type of the target variable more specific than the static type of the source variable.

Casting operator

?= or MOVE ... ?TO … .

Abstract Classes and Methods

When needed a class model for subclasses and do not require any objects from this class.

Abstract Class

Syntax:

CLASS class DEFINITION ABSTRACT.

...

ENDCLASS

Abstract Method

```
CLASS class DEFINITION ABSTRACT.
  PUBLIC SECTION.
      METHODS: method ABSTRACT.
ENDCLASS.
```

Final Classes and Methods

When needed to protect a class or individual method from uncontrolled specialization

Final Class

Syntax:

CLASS class DEFINITION FINAL.

...

ENDCLASS

Final Method

Syntax:

```
CLASS class DEFINITION FINAL.
  PUBLIC SECTION.
        METHODS: method FINAL.
ENDCLASS
```

Constructors in Inheritance

Must always take the definition of the superclasses into account

```
METHOD constructor.
  CALL METHOD super->constructor.
ENDMETHOD.
```

Interface

- Describes the public visibility section of a given class without itself implementing functionality.

- Are defined either locally or globally in the class library.

Define Interface

```
INTERFACE intf.
    DATA …
    CLASS-DATA ..
    METHODS …
    CLASS-METHODS …
    EVENTS …
    CLASS-EVENTS…
ENDINTERFACE.
```

Implement Interfaces in Classes

```
CLASS class DEFINITION.

    PUBLIC SECTION.

    …

    INTERFACES: intf1, intf2 …

    …

ENDCLASS.
```

Interfaces are incorporated solely in the public visibility section of a class.

Interface Implementation in Classes

Class must implement all the methods of all incorporated interfaces in its implementation part.

```
CLASS class IMPLEMENTATION.
    …
    METHOD intf1~ method…
        …
    ENDMETHOD.
ENDCLASS.
```

Composing Interfaces


```
INTERFACE intf1.
    METHODS meth.
ENDINTERFACE.
 INTERFACE intf2.
    INTERFACES intf1.
    METHODS meth.
ENDINTERFACE.
```

Composing Interfaces


```
CLASS class DEFINITION.
    PUBLIC SECTION.
        INTERFACES intf2.
ENDCLASS.
CLASS class IMPLEMENTATION.
    METHOD intf1~meth. … ENDMETHOD.
    METHOD intf2~meth….. ENDMETHOD.
ENDCLASS.
```

Alias Names for Interface Component

Syntax

```
CLASS class DEFINITION.
  PUBLIC SECTION.
    INTERFACES intf.
      ALIASES name FOR intf~comp.
      ...
ENDCLASS.
```

Alias Names for Interface Component

Example:

```
CALL METHOD oref~intf->method.

    =

CALL METHOD oref->alias.
```

Friend

The additions FRIENDS of statement CLASS define other classes or interfaces as friends of the class and thus grant them access to their protected and private components.

Friend

The additions FRIENDS of statement CLASS define other classes or interfaces as friends of the class and thus grant them access to their protected and private components.

Friend - Example


```
INTERFACE i1.
    ...
ENDINTERFACE.

CLASS c1 DEFINITION CREATE PRIVATE FRIENDS i1.
  PRIVATE SECTION.
    DATA a1 TYPE c LENGTH 10 VALUE 'Class 1'.
ENDCLASS.

CLASS c2 DEFINITION.
  PUBLIC SECTION.
    INTERFACES i1.
    METHODS m2.
ENDCLASS.
```

Friend - Example

```
CLASS c2 IMPLEMENTATION.
  METHOD m2.
    DATA oref TYPE REF TO c1.
    CREATE OBJECT oref.
    WRITE  oref->a1.
  ENDMETHOD.
ENDCLASS.
```

ABAP Objects:

ABAP Workbench Tool

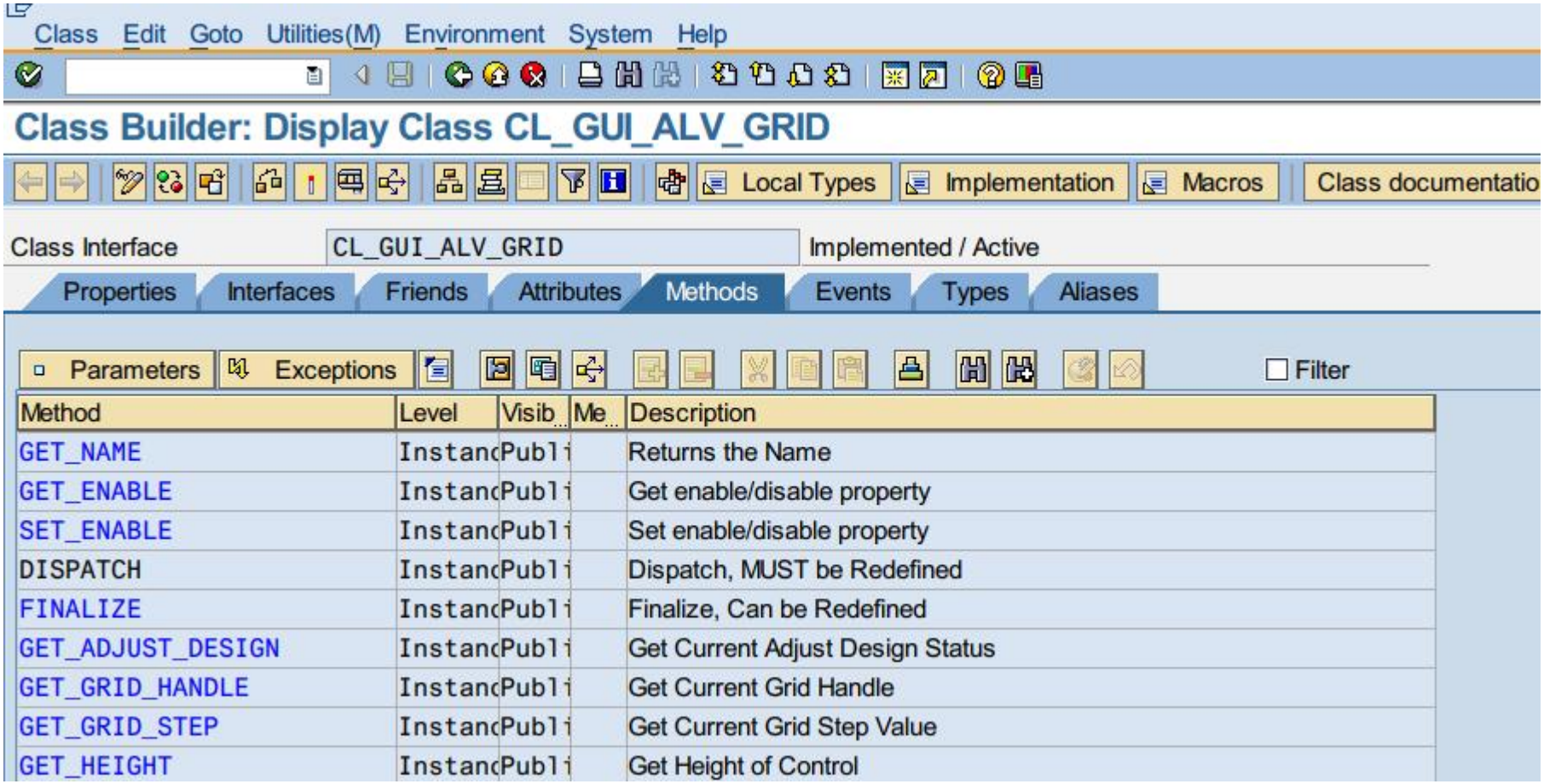
Class Builder

Global class is created with the Class Builder tool and visible in any ABAP program.

Class Builder is a special maintenance tool in ABAP Workbench

Class Builder

TCode: SE24 / SE80



Use of ABAP Objects:

OO Transaction

OO Transactions

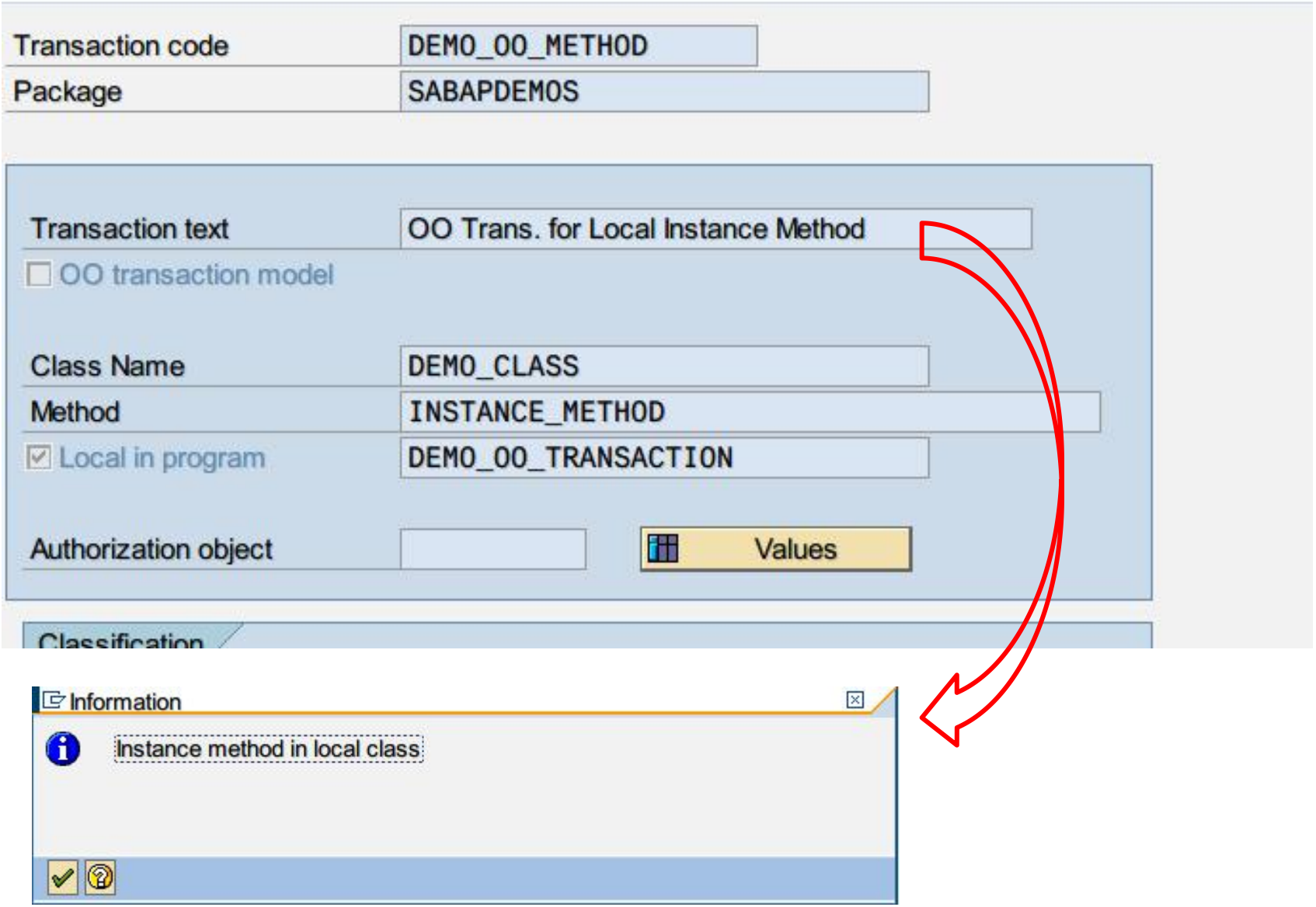
In transaction maintenance (SE93), you can specify a transaction code as an OO transaction.

OO Transactions

Mod. Pool DEMO_OO_TRANSACTION Active

```
1  *&---------------------------------------------------------------------*
2  *& Modulpool         DEMO_OO_TRANSACTION
3  *&
4  *&---------------------------------------------------------------------*
5
6  program  DEMO_OO_TRANSACTION.
7
8  *
9
10 class DEMO_CLASS definition.
11   public section.
12     methods INSTANCE_METHOD.
13 endclass.
14
15 *
16
17 class DEMO_CLASS implementation.
18   method INSTANCE_METHOD.
19     message 'Instance method in local class' type 'I'.
20   endmethod.
21 endclass.
22
```

OO Transactions



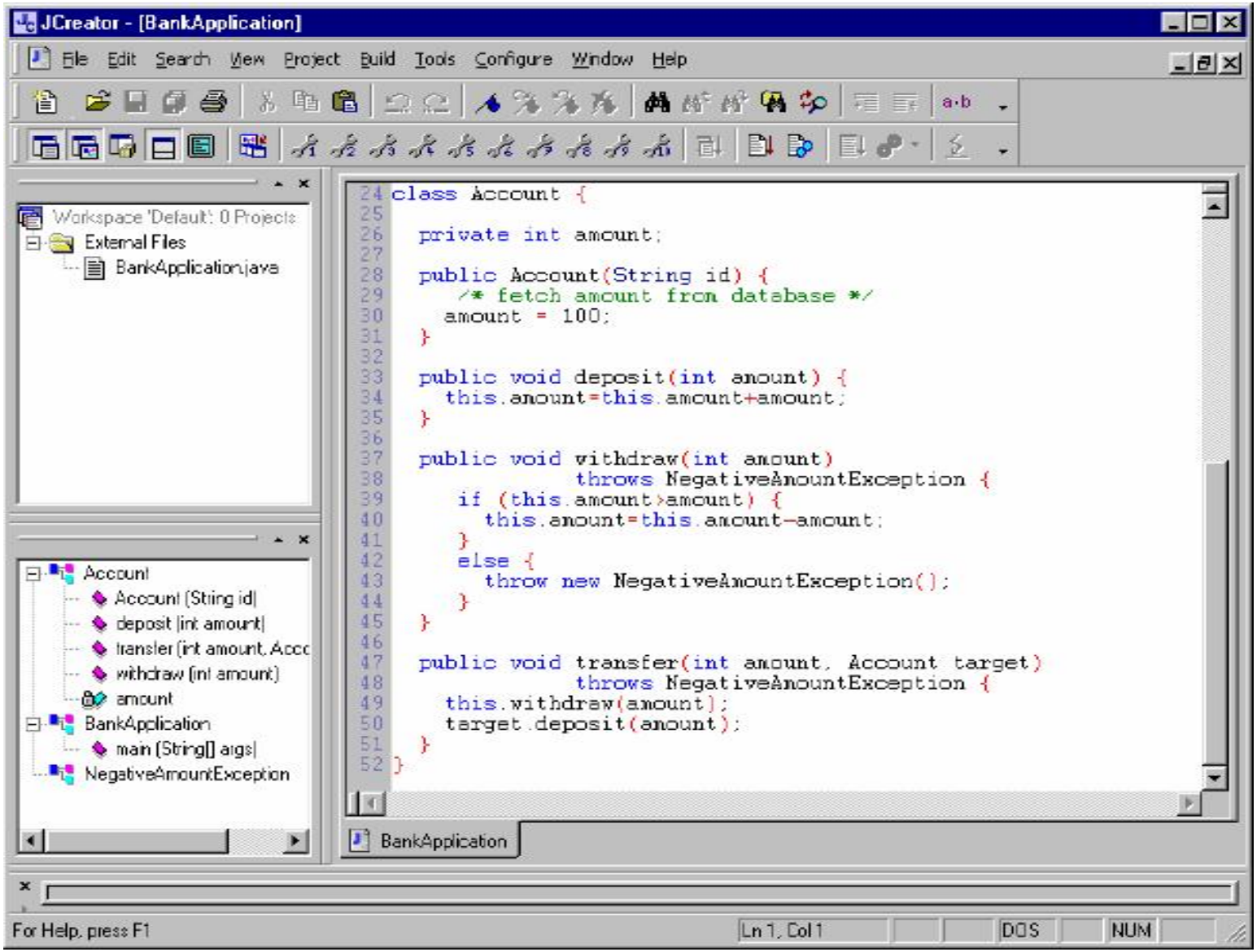
ABAP Objects for Java Programmer

Similarities Between Java and ABAP

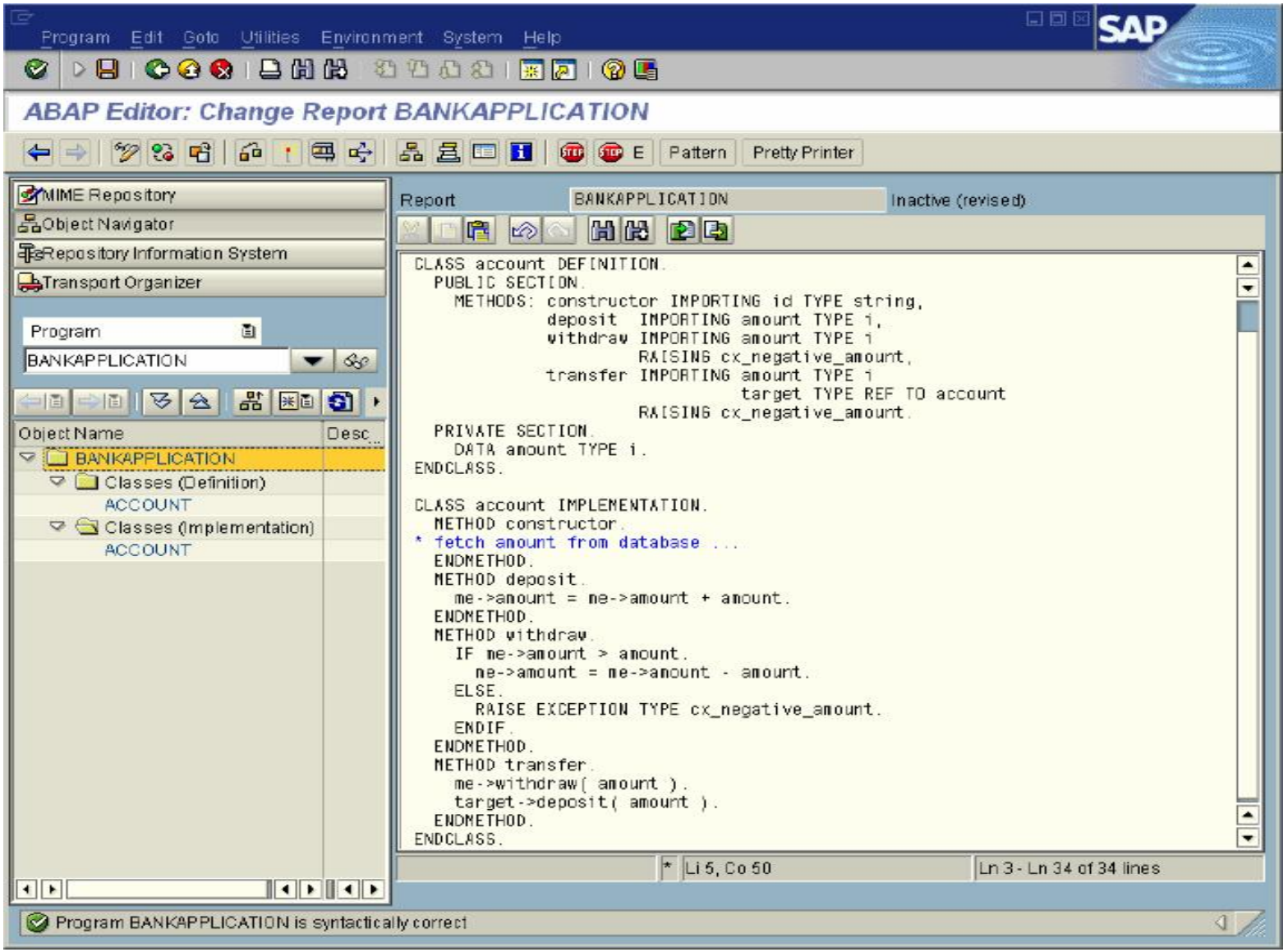
Like Java,

- **ABAP is a programming language**
- **ABAP is object-oriented**
- **ABAP runs on a virtual machine**
- **ABAP is the foundation for a whole technology**

Similarities - Typical Java Development Tool



Similarities – ABAP Development Tool



Similarities - Classes

```
class Account {

  private int amount;

  public Account(String id) {
    ...
  }

  public void deposit(int amount) {
    ...
  }
  ...
}
```

Java

```
CLASS account DEFINITION.
  PUBLIC SECTION.
    METHODS:
      constructor IMPORTING id TYPE string,
      deposit IMPORTING amount TYPE i,
  PRIVATE SECTION.
    DATA:
      amount TYPE i.
ENDCLASS.

CLASS account IMPLEMENTATION.
  METHOD constructor.
    ...
  ENDMETHOD.
  METHOD deposit.
    ...
  ENDMETHOD.
  ...
ENDCLASS.
```

ABAP

Syntactical differences aside, ABAP supports classes as Java does

In ABAP, the declaration and implementation of a class are split into separate parts

Similarities - Classes

```
private int amount;
```

Java

```
PRIVATE SECTION.
  DATA:
    amount TYPE i
```

ABAP

```
public void transfer(int amount,
                     Account target)
       throws NegativeAmountException
```

Java

```
PUBLIC SECTION.
  METHODS:
    transfer IMPORTING
               amount
               target TYPE REF TO account
             RAISING cx_negative_amount.
```

ABAP

Syntactical differences aside, ABAP supports attributes and methods as Java does (apart from method name overloading)

In ABAP, a class has three visibility sections (PUBLIC, PROTECTED, PRIVATE)

Similarities – Method Implementations

```
public void withdraw ... {
      if (this.amount>amount) {
        this.amount=this.amount-amount;
      }
      else {
        throw new NegativeAmountException();
      }
  }
```

Java

```
METHOD withdraw.
    IF me->amount > amount.
      me->amount = me->amount - amount.
    ELSE.
      RAISE EXCEPTION TYPE cx_negative_amount.
    ENDIF.
ENDMETHOD.
```

ABAP

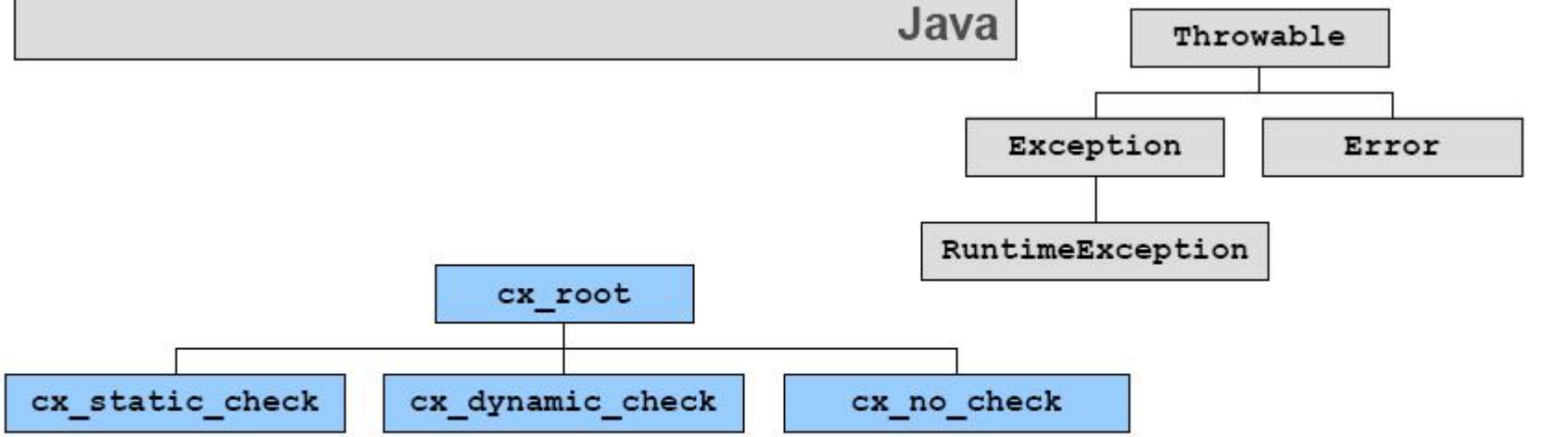
For basic operations, syntax can be translated almost one to one

- ”->” (or ”-”) replaces ”.”, ”.” replaces ”;”, ”me” replaces ”this”
- In ABAP, blocks are built by keywords instead of {}
- Raising exceptions follows the same principles

Similarities – Exceptions and Inheritance

```
class NegativeAmountException extends Exception{ }
```

Java



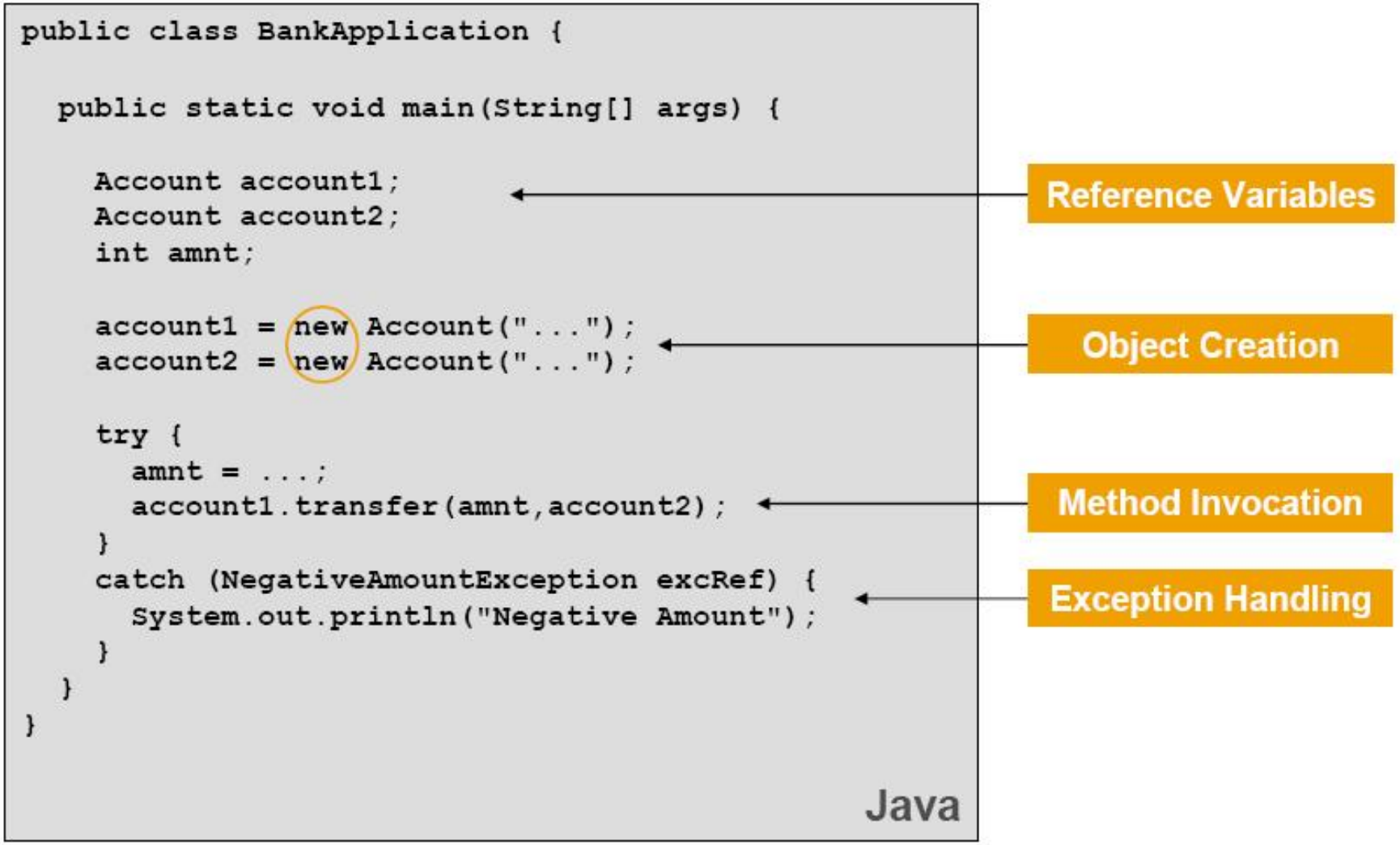

```
CLASS cx_negative_amount DEFINITION INHERITING FROM cx_static_check.
ENDCLASS.
```

ABAP

Exception classes must be derived from predefined superclasses

The superclasses define how exception handling is checked

Similarities – Handling Objects in Java



Similarities – Handling Objects in ABAP

```
CLASS bank_application DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS main.
ENDCLASS.

CLASS bank_application IMPLEMENTATION.
  METHOD main.
    DATA: account1 TYPE REF TO account,
          account2 TYPE REF TO account,
          amnt     TYPE i,
          exc_ref  TYPE REF TO cx_negative_amount,
          text     TYPE string.
    CREATE OBJECT: account1 EXPORTING id = `...`,
                   account2 EXPORTING id = `...`.
    TRY.
        amnt = ...
        account1->transfer( EXPORTING amount = amnt
                                      target = account2 ).
      CATCH cx_negative_amount INTO exc_ref.
        text = exc_ref->get_text( ).
        MESSAGE text TYPE 'I'.
    ENDTRY.
  ENDMETHOD.
ENDCLASS.
```

ABAP

Reference Variables

Object Creation

Method Invocation

Exception Handling

Similarities – Keywords 1

Java	ABAP	Purpose
abstract	ABSTRACT	Abstract classes and methods
boolean	-	Boolean data type
break	EXIT	Leaves an iteration
byte	b	One-byte integer
case	WHEN	Branching
catch	CATCH	Exception handling
char	c, string	Character data type
class	CLASS	Class definition
const	CONSTANTS	Constant data object
continue	CONTINUE	Proceed with next iteration
default	WHEN OTHERS	Branching
do	DO	Iteration
double	f	Double precision floating point numbers
else	ELSE, ELSEIF	Branching
false	-	Boolean value
final	FINAL	Final classes and methods
finally	CLEANUP	Exception handling
float	-	Simple precision floating point numbers
for	WHILE	Iteration
if	IF, ENDIF	Branching
implements	INTERFACES	Interface implementation
import	-	Includes packages
int	i	Four-byte integer

Similarities – Keywords 2

Java	ABAP	Purpose
interface	INTERFACE	Interface definition
long	-	Eight-byte integer
native	BY KERNEL MODULE	Invocation API
new	CREATE	Object instantiation
null	CLEAR, INITIAL	Null reference
package	- (supported by tools)	Packages
private	PRIVATE	Private components
protected	PROTECTED	Protected components
return	RETURN	Leaves a method
short	s	Two-byte integer
static	CLASS-...	Static class components
strictfp	-	Enforces IEEE norm for floating point operations
super	super	Addresses the super class
switch	CASE	Branching
synchronized	CALL FUNCTION DESTINATION IN GROUP	Threading/parallel processing
this	me	Self reference
throws	RAISING	Propagates exceptions
transient	- (implemented by self defined serialization)	Prevent serialization
true	-	Boolean value
try	TRY	Exception handling
void	RETURNING	Return value of a method
while	WHILE	Iteration

Use of ABAP Objects:

The Control Framework

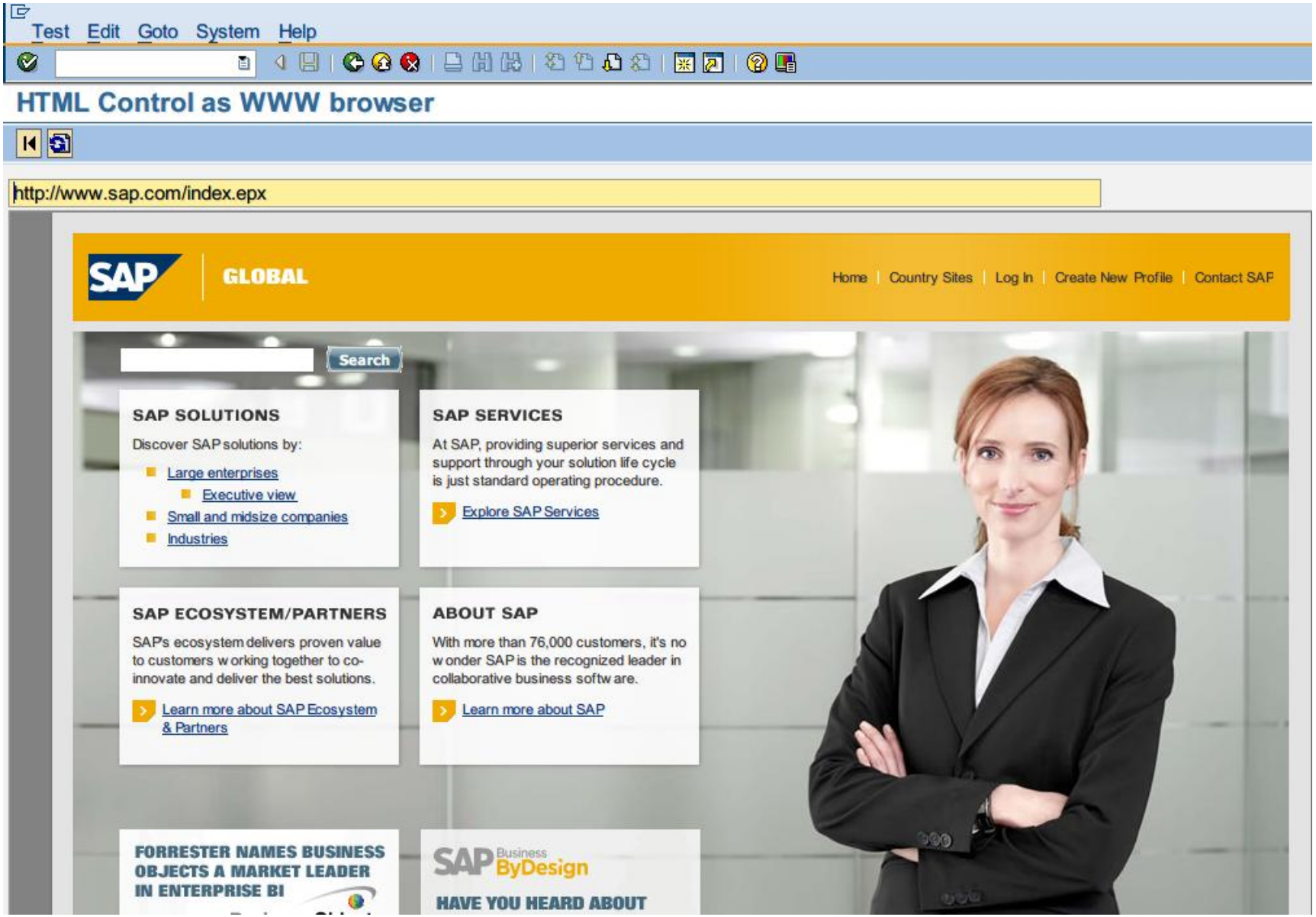
The Control Framework (CFW)

In the SAP system, you can use ABAP to control desktop applications (custom controls).

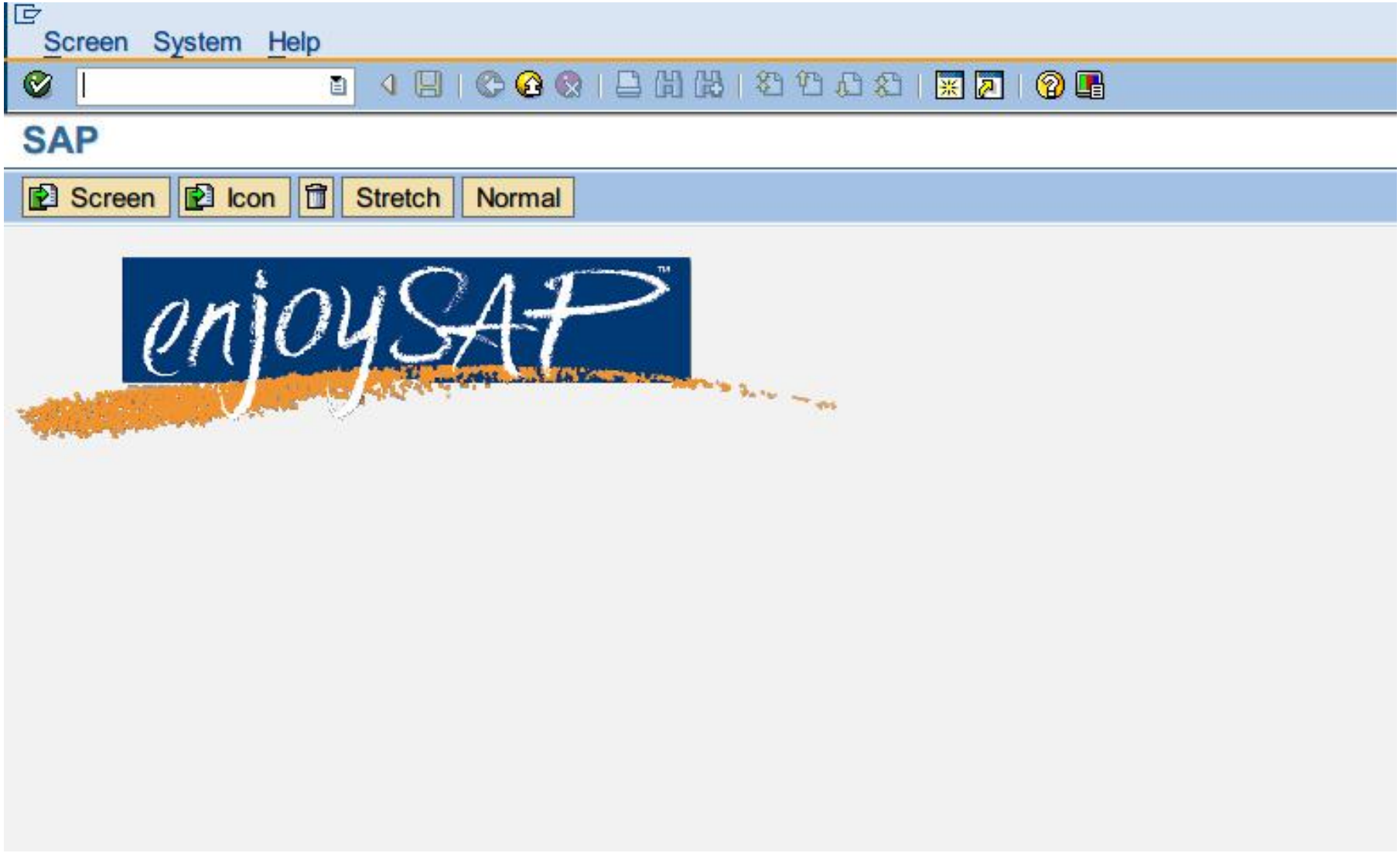
The application logic runs on the SAP application server, which drives the custom control at the frontend.

The Control Framework supports controls (ActiveX and JavaBeans) that are implemented within the SAP GUI.

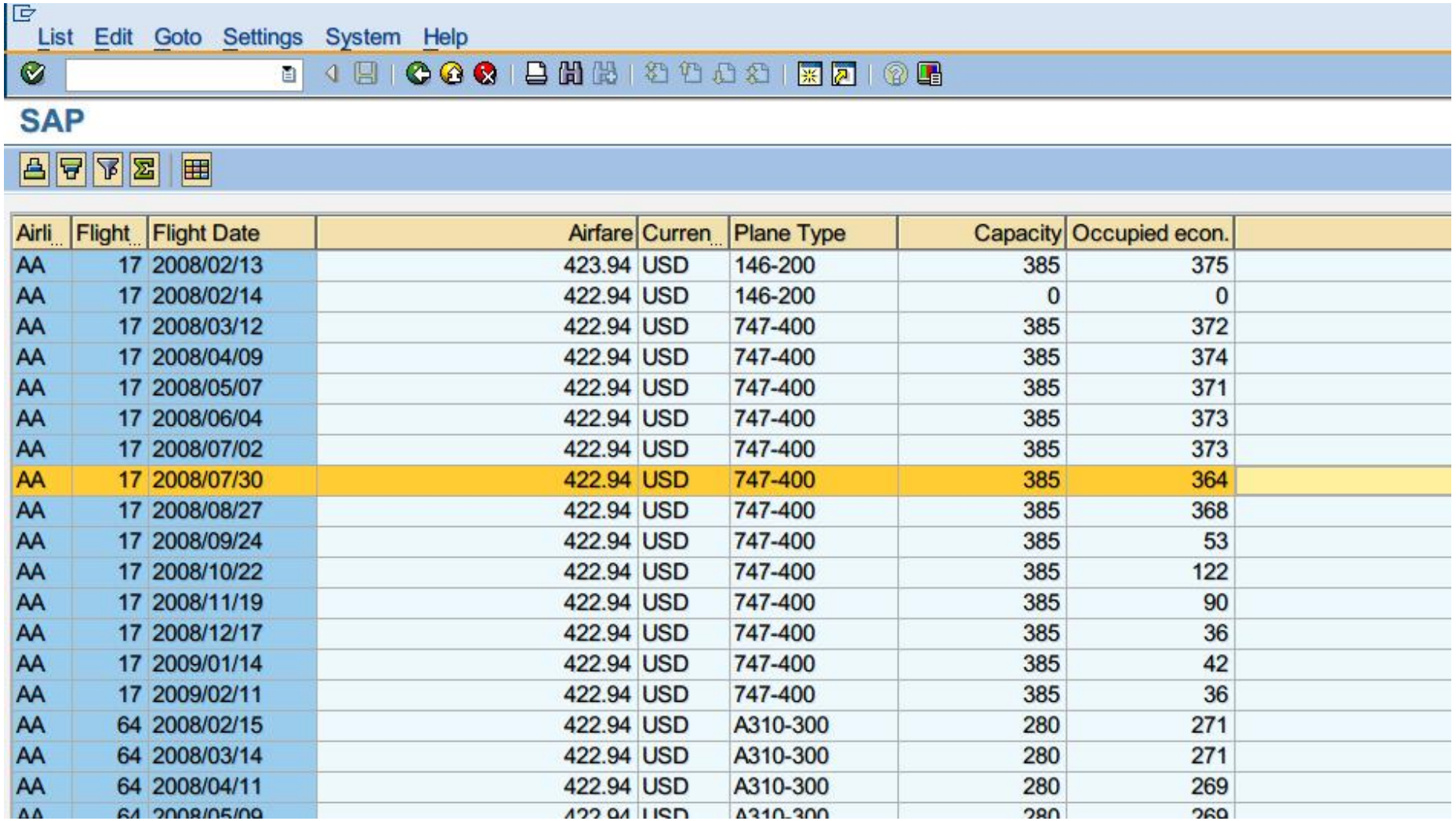
CFW – HTML Control



CFW – Picture Control



CFW – ALV Grid Control



Airli...	Flight...	Flight Date	Airfare	Curren...	Plane Type	Capacity	Occupied econ.
AA	17	2008/02/13	423.94	USD	146-200	385	375
AA	17	2008/02/14	422.94	USD	146-200	0	0
AA	17	2008/03/12	422.94	USD	747-400	385	372
AA	17	2008/04/09	422.94	USD	747-400	385	374
AA	17	2008/05/07	422.94	USD	747-400	385	371
AA	17	2008/06/04	422.94	USD	747-400	385	373
AA	17	2008/07/02	422.94	USD	747-400	385	373
AA	17	2008/07/30	422.94	USD	747-400	385	364
AA	17	2008/08/27	422.94	USD	747-400	385	368
AA	17	2008/09/24	422.94	USD	747-400	385	53
AA	17	2008/10/22	422.94	USD	747-400	385	122
AA	17	2008/11/19	422.94	USD	747-400	385	90
AA	17	2008/12/17	422.94	USD	747-400	385	36
AA	17	2009/01/14	422.94	USD	747-400	385	42
AA	17	2009/02/11	422.94	USD	747-400	385	36
AA	64	2008/02/15	422.94	USD	A310-300	280	271
AA	64	2008/03/14	422.94	USD	A310-300	280	271
AA	64	2008/04/11	422.94	USD	A310-300	280	269
AA	64	2008/05/09	422.94	USD	A310-300	280	269

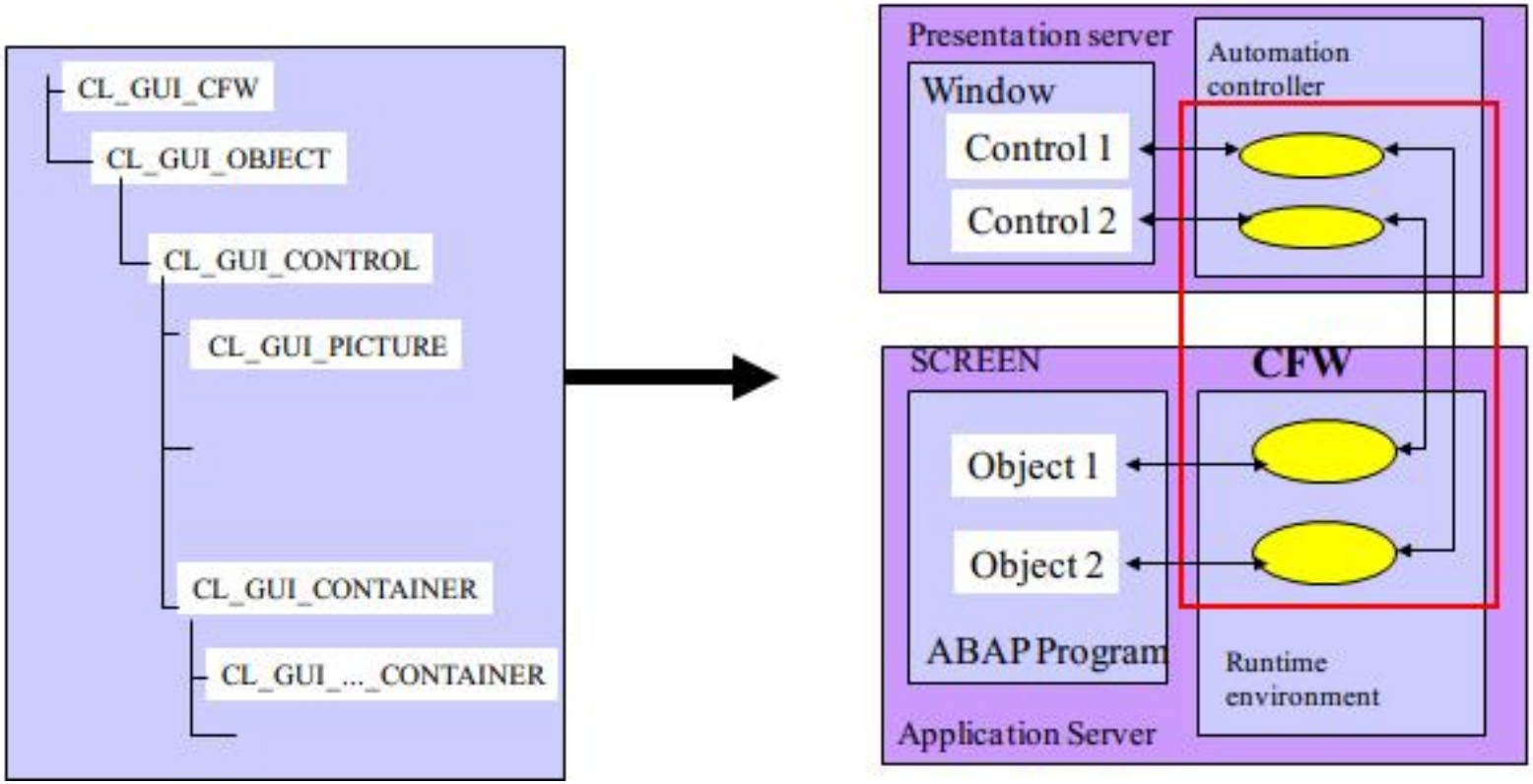
Control Framework Architecture

Global class in the class library for each GUI control supplied and that when working with GUI controls, for each control object on the frontend there is a proxy object of the corresponding class in the ABAP programs.

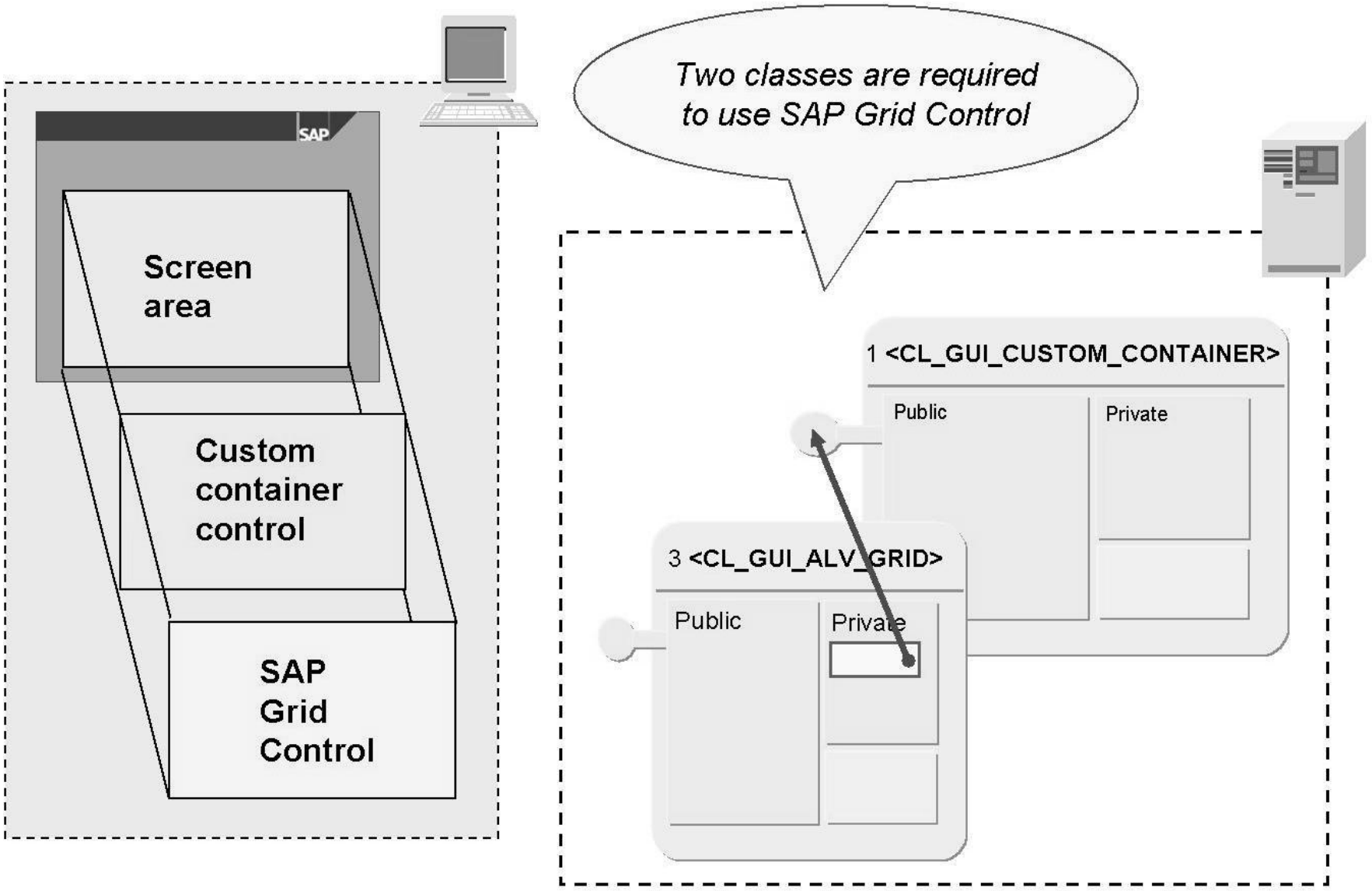
Control Framework Architecture

ABAP programming works only with the proxy objects

The CFW forwards its method calls to actual GUI control objects and informs the objects of the ABAP program by triggering events about the user actions on the GUI controls.



Control Framework Architecture

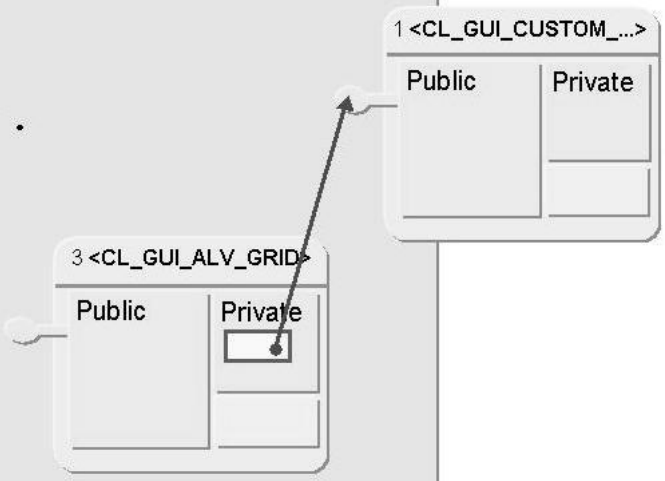


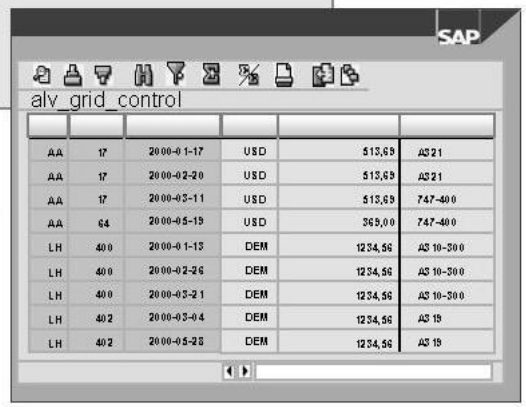
Control Framework Example

```
...
DATA: container_r TYPE REF TO cl_gui_custom_container,
      grid_r      TYPE REF TO cl_gui_alv_grid.
...
CREATE OBJECT container_r
    EXPORTING container_name = 'CONTAINER_1'.

...
CREATE OBJECT grid_r
    EXPORTING i_parent = container_r.
...

grid_r->set_table_for_first_display(
            i_structure_name = 'SAPLANE'
            it_outtab        = itab_saplane ).
...
```





Container Controls

CL_GUI_CUSTOMER_CONTAINER

CL_GUI_DOCKING_CONTAINER

CL_GUI_SPLITTER_CONTAINER

CL_GUI_DIALOGBOX_CONTAINER

Application Controls

CL_GUI_TOOLBAR

CL_GUI_PICTURE

CL_GUI_HTML_VIEWER

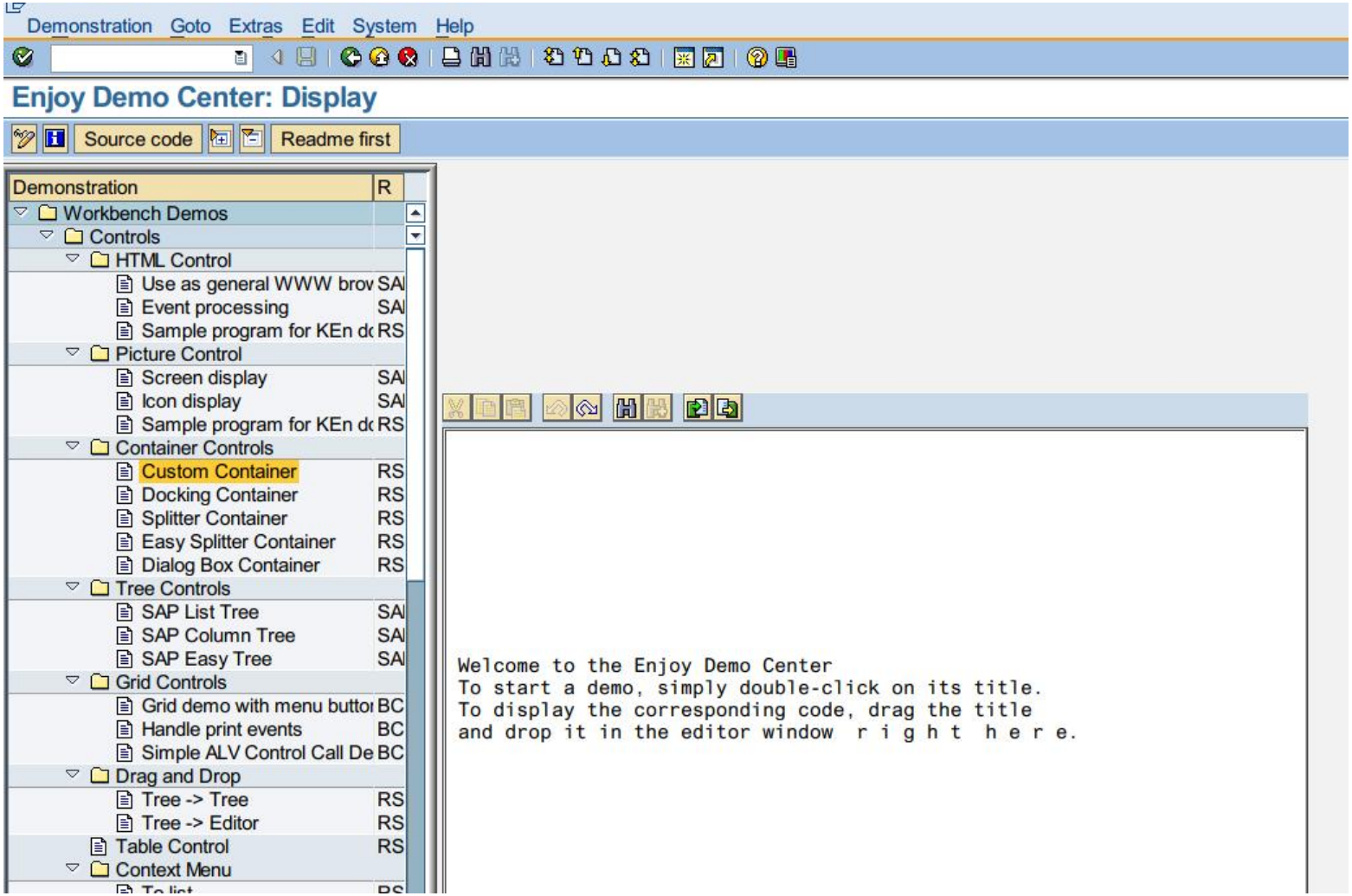
CL_GUI_TEXTEDIT

CL_GUI_SIMPLE_TREE

CL_GUI_ALV_GRID

Enjoy Demo Center

TCode: DWDM



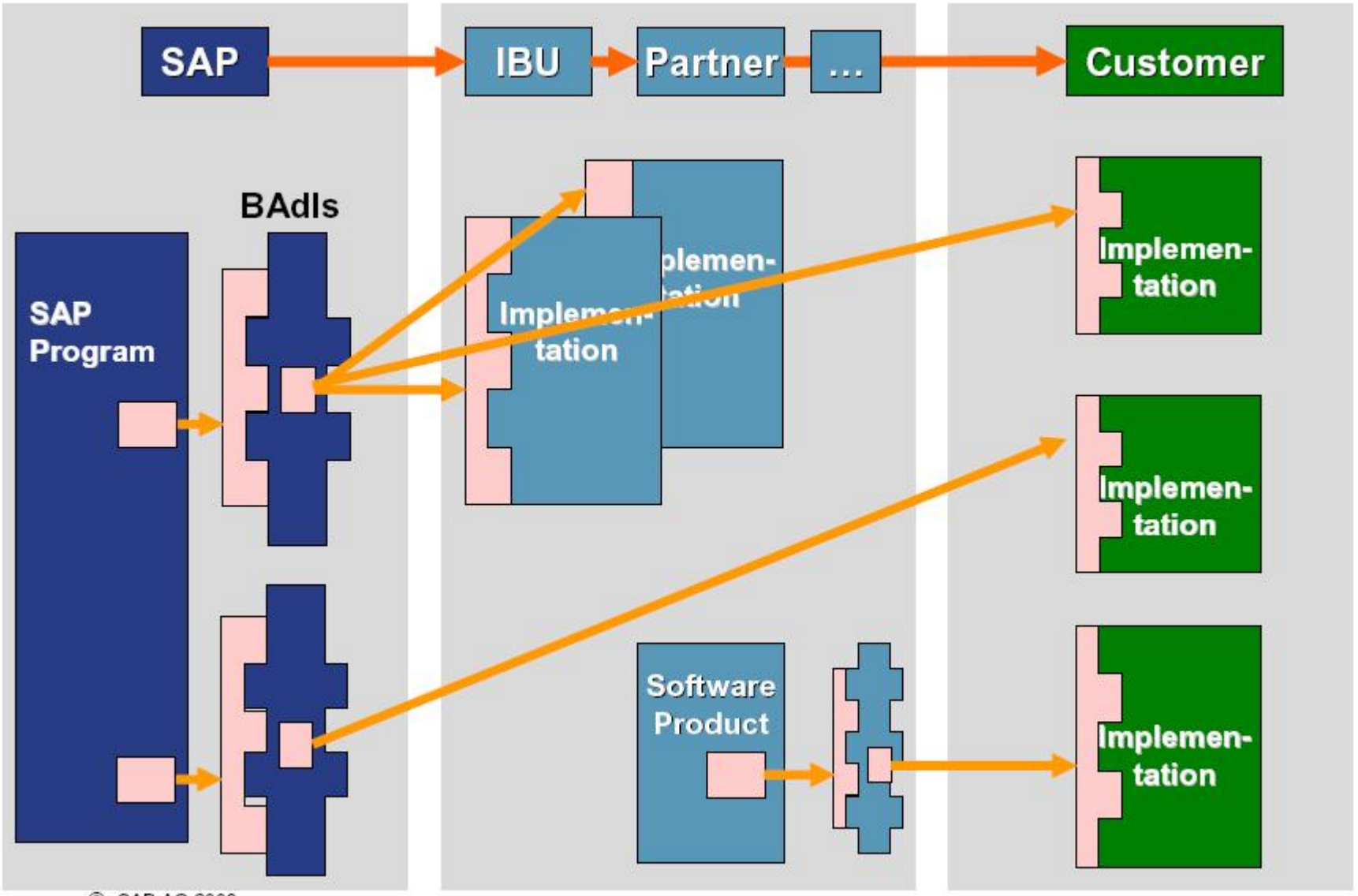
Use of ABAP Objects:

BAdIs

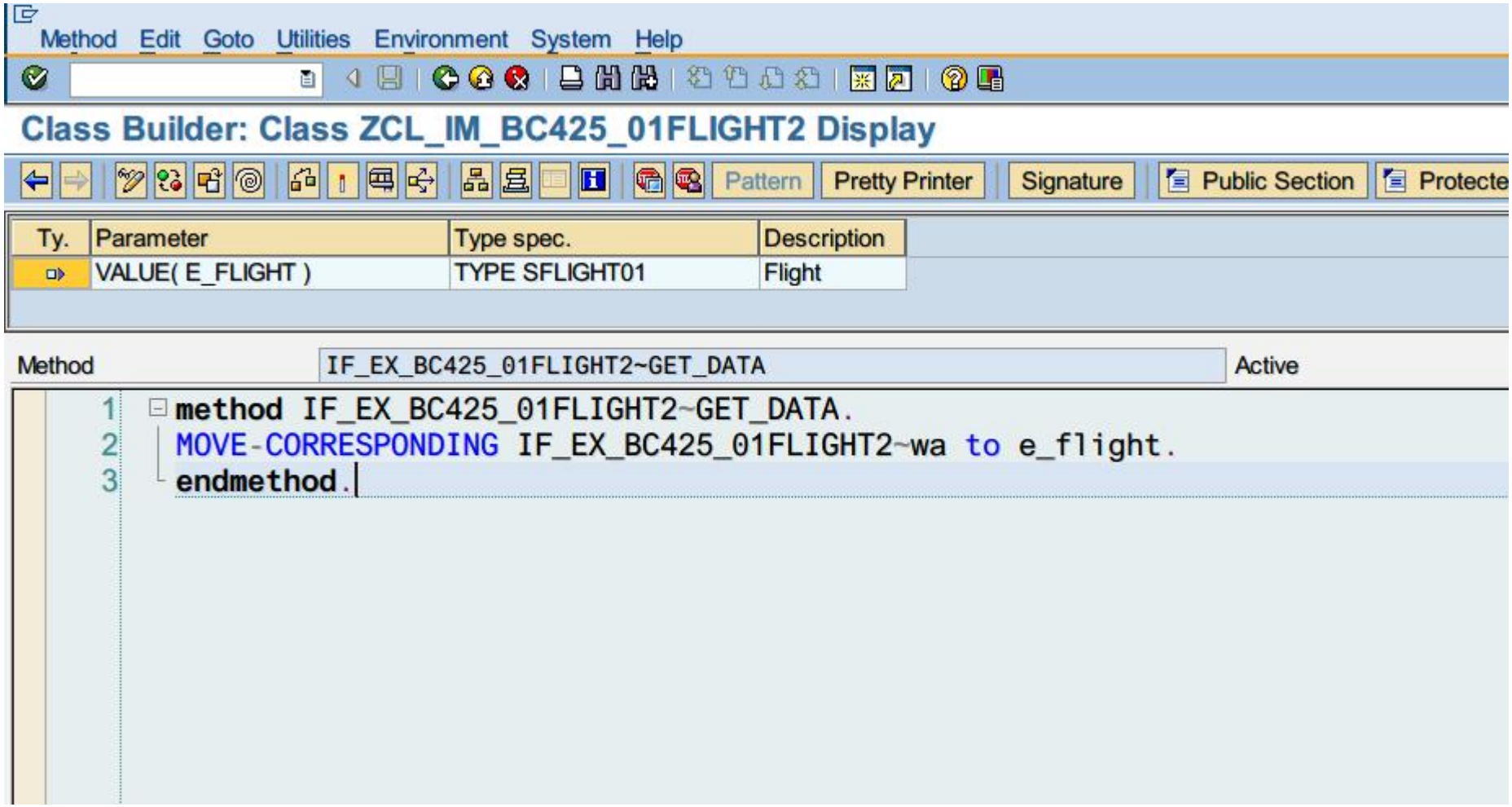
Enhancement using Business Add-Ins

Business Add-Ins (BAdIs) are one of the most important technologies used to adapt SAP software to specific requirements.

Business Add-Ins: Architecture



Implementation of Business Add-Ins



Use of ABAP Objects:

Web Programming

Web Programming

- BSP
- Web Dynpro for ABAP
- Custom HTTP Handler Class
- …

Q & A

Copyright 2008 SAP AG. All Rights Reserved