

# ABAP Objects: Best of C++, Java and COM?

Jürgen Heymann  
SAP AG



## Overview

- **The ABAP Language Environment**
- **ABAP Objects Concepts Overview**
- **Discussion of some Design Decisions**
- **Questions and Answers**



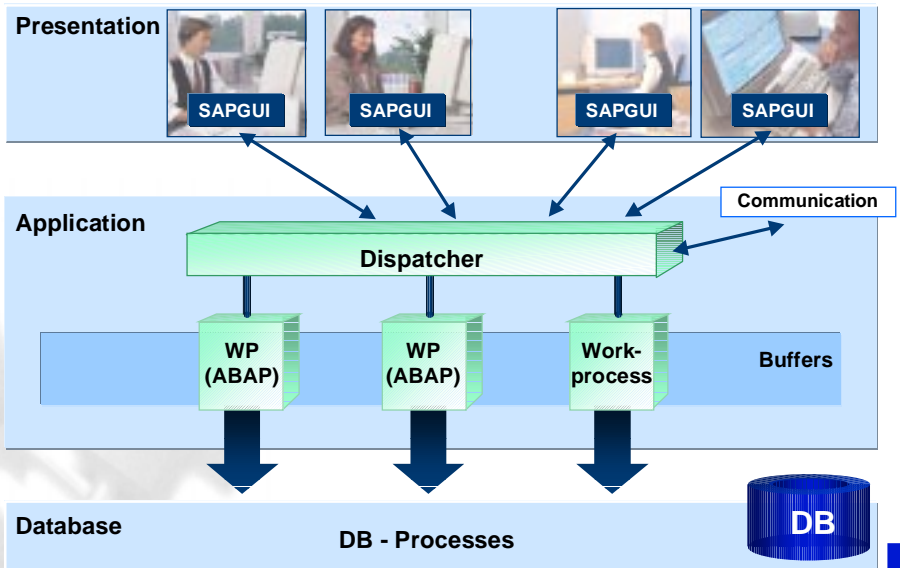
# The ABAP Language Environment

**Q: *What are goals and constraints of designing the OO extensions to ABAP/4?***

- **Extreme Variability**
  - R/3 is Framework / CORE Application
  - Customization
  - 5-levels of Extensions (layered)
  - Release-Cycles of individual system parts



## R/3 Basic Architecture



# Scale

- > 18 000 DB Tables + Views
- > 15 000 Programs
- > 35 Mio Source Lines (10 Mio comment lines)
- > 58 000 Functions (central)
- > 55 000 Screens
- DDIC: > 100 000 Data Elements + Domains,  
> 20 000 Structures
- > 2 GB LOAD Size (compiled system)



# ABAP History

- Started as macro language
- R/2 ==> R/3 migration (mostly compatible)
- Very good integration with database
  - SELECT INTO, SELECT LOOP in ABAP
  - intelligent buffering, array fetch
- Geared towards processing of tables (relational)
  - powerful table statements
  - key tables, sorted tables



# Why Objects at SAP?

- **BOR (Business Object Repository) was already there**
- **We wanted OO benefits within the system**
  - **Primary problems: complexity and maintainability (inherent and technical complexity)**
  - **=> encapsulation, reuse, abstraction mechanisms**
  - **Basis for patterns and frameworks**
- **Basis for better interoperability with DCOM/CORBA**
- **Basis for new GUI features: Controls & Control Framework**
- **OO can use existing software infrastructure**



# Design Goals for ABAP Objects

- **simple (ASAP, simpler than Java?)**
- **100% upwards compatible with ABAP/4, integration**
- **'main stream', use proven OO concepts**
- **comply with important external standards (DCOM/CORBA, Java/VB/...), UML**
- **target application-programming, not systems-programming**
- **efficient implementation**
- **usable for SAP environment: distributed development**



# A Coding Sample

```
CLASS Ctruck DEFINITION.  
  PUBLIC SECTION.  
    DATA: VehicleId TYPE I READ-ONLY.  
    METHODS: LoadParcel IMPORTING Parcel TYPE REF TO CParcel,  
             UnloadParcel ...  
  PRIVATE SECTION.  
    DATA: ParcelTab TYPE REF TO CParcel OCCURS 0.  
ENDCLASS.  
  
CLASS Ctruck IMPLEMENTATION.  
  METHOD LoadParcel.  
    APPEND Parcel TO ParcelTab.  
    "-- do more stuff ...  
  ENDMETHOD.  
ENDCLASS.
```

```
DATA truck1 TYPE REF TO Ctruck,  
    ...  
    "-- get input data for parcel from somewhere ...  
CREATE OBJECT parcel.  
CALL METHOD parcel->SetPars EXPORTING Weight = In_weight.  
    "-- deal with multiple instances  
CALL METHOD truck1->UnloadParcel IMPORTING Parcel = Parcel.  
CALL METHOD truck2->LoadParcel( Parcel ).
```

- The ABAP Language Environment
- ➔ [ABAP Objects Concepts Overview](#)
- Discussion of some Design Decisions
- Questions and Answers

# ABAP Objects Concepts

- Classical Object Model (like C++, Java, ...)
- Attributes, methods and events (instance and class-level)
- Single inheritance for classes
- Interfaces, nested interfaces
- Integration with GUI and persistence
- No method overloading
- Garbage collection
- Hybrid language



- The ABAP Language Environment
- ABAP Objects Concepts Overview
- ➔ Discussion of some Design Issues
- Questions and Answers



## Some Design Issues...

- Objects also as values or only with identity?
- Which components can be defined?
- Visibility and name spaces
- Constructor et. al.
- Interfaces or multiple inheritance?
- Interfaces: Composition or inheritance/'merge'?
- Interfaces: Which components can they contain?
- Inheritance: More variations and issues



## Objects as values or only with identity?

- Issues
  - Specify at definition or use of a class?
  - So far all value semantics in ABAP/4 (values copied / compared)
- Only objects with identity (e.g. like Java)?
  - (+) simple: objects always have reference semantics (access only via reference, only references are copied / compared)
  - (+) appropriate for medium to large grained objects; logical objects always have identity
  - (-) CREATE OBJECT always needed (possibly implicit)



## Objects as Values ...(2) ?

- Also value classes (ADTs, like C++)?
  - (-- Copy constructors, value contexts: IF (\*), 'compare', copy (deep?), value(), initializer lists for constructors
  - (-) 'containment' (aggregation) often not implementable by 'embedded object'
  - (-) Value classes like String, TimeStamp integrated
  - (-) Polymorphism only on references; 'compatibility' of structures very different, cut-off when assigning subclass value to superclass variable?



## Objects as Values ...(3) ?

- Resolution (for SAP)
  - Objects have reference semantics, i.e. 'sharing' only for objects
  - Separate value classes possibly later



# Which Components can be defined?

- **Attributes, constants and methods: no question**
- **Class (static) Attributes and methods: no question**
- **Events integrated (publish-subscribe). Why?**
  - Clumsy as 'add-on' (parameter, relies on inheritance), events have parameters
  - For loose / inverted coupling of subsystems as well as GUI
  - Handler methods definable (signature comes from event)
  - Explicit SET HANDLER (usable for GUI and workflow)
  - Mass registration



# Visibility and Name Spaces

- **SECTIONS for visibility instead of per component**
  - Class user sees only what is 'public'
  - More efficient for compiler, less recompilation
  - extensible: MAPPING Section (DB)
- **Name space for components**
  - All components of classes and interfaces in one name space
  - simple rule
  - for DCOM/CORBA mapping (otherwise 'external names')



# Constructor et. al.

- **Constructor definable**
  - not needed for static initialization of attributes
- **No destructor!!!**
  - no need to free storage (garbage collection)
  - time of call completely undefined, problems of ordering
  - no destructor calls at end of transaction
  - only to manage external resources, but there not safe. Instead manage via RFC connection
- **Class constructor when class gets 'first used'**
  - to initialize class attributes
  - no class destructor



# Multiple Inheritance or Interfaces?

- **Choices:**
  - only composition + aggregation, no inheritance (COM)
  - single inheritance + interfaces (Java, COM+, ...)
  - multiple inheritance (C++, Eiffel, ...)
  - delegation (layered objects, no common identity)
- **Goals:**
  - code reuse
  - polymorphism, multiple levels of abstraction
  - the goals can be met independently!



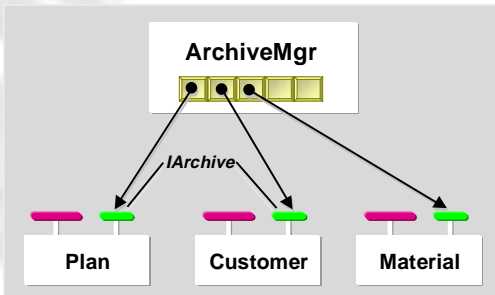
# Multiple Inheritance or ... (2)?

- **Problems:**
  - mostly used as 'mixin' = interface
  - polymorphism on all axes (substitutability is hard)
  - name conflicts (in distributed development!)
  - multiple base classes? (diamond inheritance)
  - combinatorial explosion for multiple classification
    - ◆ must be resolved by role pattern or the like
    - ◆ would be static, not dynamic
- => **multiple inheritance doesn't do it!**



# Interfaces...

- ...define interaction between objects
- ...offer polymorphism independent of inheritance
- Classes can implement many interfaces
- Uniform access through interface reference



# Interfaces: Composition or 'merge'?

- **Merge / Inheritance (Java):**
  - all components of all interfaces 'meet' at the enclosing interface / at the implemented class
  - conflict resolution by overloading for methods
  - => only methods in interfaces
  - **Problem: semantic conflicts cannot be resolved!**

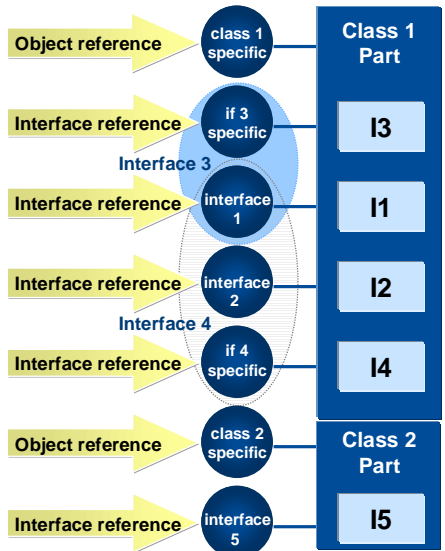
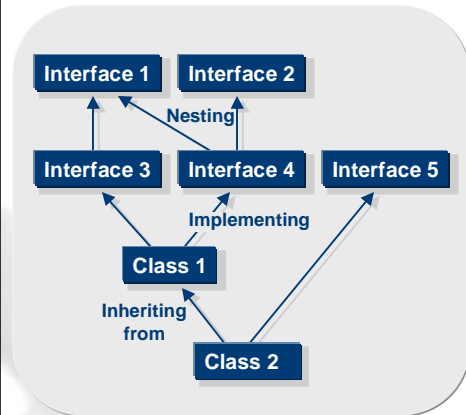


# Interfaces: Composition ... (2)

- **Our Model: Because of: distributed development, semantic conflicts, no overloading ...**
  - => Interface Composition (nesting), implemented side-by-side (COM)
  - Components of nested / implemented interfaces not visible on 'top level'. Access by navigation "i = o", "i1 = i2".
  - **ALIASES: make components of nested interfaces visible in the enclosing interface => can always define new clean abstractions, no name conflicts**
  - **no automatic aliases because of possible name conflicts !!!**



# Interfaces and Classes



## Interfaces: Which Components ...

- Which components can interfaces have?
    - Methods, events, constants (and static ...): no question
  - Attributes?
    - Pro:
      - ◆ Class = Interface w.r.t. component definition
      - ◆ You can 'pull off' interface from 'public interface' of class
      - ◆ No (name) conflicts for attributes since they are implemented side-by-side
      - ◆ Attributes can be protected as READ-ONLY
- Resolution: Same components in interfaces as in classes

# Why no Overloading?

- Method Name Overloading: distinguish by signature
- Purpose?
  - Polymorphism
  - Resolve name conflicts (when merging name spaces)
- Why not?
  - What about attributes (also in interfaces)?
  - Yet another polymorphism (object references, interfaces)
  - Doesn't solve semantic conflicts (e.g. 2 print(int) methods in 2 interfaces in Java)

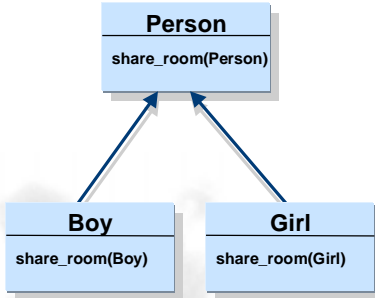


# Inheritance: More Variations and Issues

- Is difficult to use
  - Essence is **substitutability** ! Intuitive understanding is 'subsets' (e.g. rectangular <-> square? container problems, car => red\_cars)
  - 'is\_a'? 'is\_kind\_of' (transitive) is not 'is\_instance\_of' (not transitive): 'sparky is\_a terrier', 'terrier is\_a dog\_breed', but not 'sparky is a dog\_breed'
  - inheritance used wrong 60-80% in C++ projects (and text-books): violating substitutability
  - 'covariance'? narrowing the capabilities of a subclass



# Inheritance: Covariance Problem



```
p : Person;
b : new Boy();
g : new Girl();

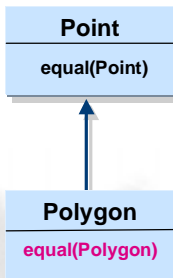
p = b;
p.share_room(g); // runtime error!
```

→ You can only have 2 of these 3:

- static type safety
- substitutability
- covariance



# Inheritance: Covariance II



<== Syntax error??

## ● Alternatives

- resolve by method overloading  
but: use is still surprising

```
p : Point;
q : Polygon;

p.equal(q); // P::equal
q.equal(p); // syntax error!!
```



# Inheritance ...

- **Further problems**

- **Base-class must be designed for inheritance**
- **Control flow complexity, fragmentation of code**
- **extremely tight coupling to super class (distributed development!)**
- **'fragile base-class' (subclass can cause errors in super class)**
- **Testing: subclass must be tested again after every change of super class (and vice versa!)**
- **Inheritance no vehicle for extensibility (formerly so in the BOR)**



# Inheritance ...

- **But: Pro Inheritance**

- **Basis for many patterns and frameworks (clumsy with interfaces)**
- **GUI Controls, System-level code**
- **Small flat inheritance hierarchy sometimes useful (code reuse)**



# Inheritance for SAP?

- **Single inheritance for small (controlled) hierarchies, + interfaces**
  - For GUI, patterns & frameworks, less important for application programming
- **No change of signature (no-variance)**
  - promotes substitutability
- **Limit inheritance by packages**
- **No inheritance between SAP and extensions (customers, partners ...)**



# Comparison: ABAP Objects, C++, Java

	<i>ABAP Objects</i>	<i>Java</i>	<i>C++</i>
<i>Value Objects</i>	no	no	yes
<i>Components</i>	Attributes, Methods, Events	Attributes, Methods	Attributes, Methods
<i>Inheritance</i>	single	single	multiple
<i>Interfaces</i>	yes (composition)	yes (merge)	no (abstract class)
<i>Constructor...</i>	just Constructor	Constructor Destructor	Constructor Destructor



## Summary & Evaluation

- Objects integrated into ABAP (100% compatible)
- Usable for SAP environment (distributed development, SW logistics)
- DCOM/CORBA: restrictions for 'externalized' classes / interfaces
- UML is lacking a few things ... (&Design-Tools!)

A vertical graphic on the left side of the slide. It features a dark, textured background with several computer monitors or screens floating in a space-like environment. The word 'Discussion' is written vertically in large, white, sans-serif font across the graphic.

Discussion

**Questions (!)**

**&**

**Answers (?)**

