

Working with OpenEdge 10

Using object oriented language features

... in procedural code

Mike Fechner, Consultingwerk Ltd.

Object-orientation in the ABL

- Introduction
- Comparing OOABL to procedural
- Sample 1: Parameter object
- Sample 2: Static methods, simplified API
- Sample 3: Static events
- Sample 4: FINALLY Block
- Sample 5: CATCH ABL runtime error

Introduction / disclaimer

- This talk is intentionally not about object oriented concepts and architecture!
- This talk is to introduce some elements of the object oriented extensions to the ABL and how to use them with existing applications
- In a way that provides value

Object-orientation in the ABL

- Often referred as OOABL
 - Some people in PSC dislike that term. It's not a new language. But it's much shorter than the „object-oriented extensions to the ABL aka 4GL“
- Adds coding constructs like classes, object instantiation, methods, access types etc. to the ABL
- Constructs well known from (other) OO languages like C# and Java

OOABL time line

- **10.1A first implementation**, classes, objects, methods, properties (but I'm not 100% sure about that)
- **10.1B Interfaces**, USING statement, properties (when not part of 10.1A)
- **10.1C Static members, structured error-handling**, properties in Interfaces, DYNAMIC-NEW
- **10.2A GUI for .NET, garbage collection** for objects (anything reference by a WIDGET-HANDLE or COM-HANDLE is not an object)
- **10.2B Abstract classes**, abstract members, .NET generic type definition, **strong typed events**, reflection part 1

OOABL: Why?

- Common way of coding, known to every young developer. Universities don't teach procedural programming at all
- Market trend that PSC could not resist anymore, didn't want to be the only modern non OO language (together with giving up the 4GL name)
- Simply the way people want to code today. True for ABL?
- GUI for .NET! No way to leverage the .NET framework without OO

OOABL: Why?

- Different way of coding, breaking tasks into smaller chunks, agile coding, Unit-Testing, reusability (but we still believe the ABL is the most powerful language to maintain spaghetti code)
- Interfaces, Design Patterns, often quoted Guru's (not sure if everybody read their books)
- Martin Fowler (UML, OO architecture, transformation)
- GoF, Gang of Four, Erich Gamma et al., often quoted in source code (i.e. GoF 175 – decorator pattern)

OOABL: Members of a class

- Constructor(s)
- Destructor
- Methods, overloaded methods, polymorph meth
- Data Members
 - Properties
 - Variables (primitive and reference types)
 - Defined non OO objects, Temp-Tables, ProDatasets
- Events (10.2B)

Demo / Sample code

- IBusinessEntity Interface
- BusinessEntity class
- proSIretrieve.p
- Class Browser



Object-orientation in the ABL

- Introduction
- Comparing OOABL to procedural
- Sample 1: Parameter object
- Sample 2: Static methods, simplified API
- Sample 3: Static events
- Sample 4: FINALLY Block
- Sample 5: CATCH ABL runtime error

Comparing OOABL to procedural

- The compiler and runtime enforce OO concepts (but not good OO design 😊)
 - strong-typing, type safeness
- Concepts like inheritance or encapsulation are part of the language
- ADM2 implemented them in the language
- Object instance is very similar to a persistent procedure: Launched by someone else, life cycle, private and public members

Comparing OOABL to procedural

- Comparing SUPER class to SUPER procedure
- Both allow „inheriting“ behaviour by using methods and procedures of the SUPER thing
- SUPER procedure: Manual task to launch or locate that, SUPER class: Just INHERIT
- Shared SUPER procedure: Data Members (Variables, Temp-Tables, Buffer, ...) shared among childs, SUPER class: individual to every single child
- Child class won't compile with error in SUPER class

TABLE-HANDLE parameter

- A SUPER procedure expecting a TABLE-HANDLE parameter can be overloaded with a TABLE parameter
- Compiler doesn't care (know about it)
- Runtime is fine with that
- This is not possible with classes:
 - Method with TABLE-HANDLE cannot be overridden by method with TABLE parameter
 - Interface method with TABLE-HANDLE parameter cannot be implemented using TABLE

Dynamic coding

- Procedures
 - RUN VALUE („...“) . RUN VALUE („...“) IN hProc.
 - DYNAMIC-FUNCTION, {fn}, {fnarg}
 - INTERNAL-ENTRIES property
 - DYNAMIC-CALL
- OO
 - Don't do that 😊 Except DYNAMIC-NEW
 - DYNAMIC-NEW (10.1C), DYNAMIC-INVOKE (10.2B)
 - No dynamic access to properties
 - No possibility to query available members ☹

AppServer

- The AppServer protocol only speaks procedural
- Every client needs to call into procedures
- Activate, Deactivate, Connect, Disconnect procedures
- AppServer may use objects from there on
- Can't pass an object as a parameter between AppServer and Client
- Can't remotely call into an object like we can into a remote persistent procedure (not recommended, but possible)

Garbage collection

- For true objects only (Progress.Lang.Object, System.Object)
- Instance is removed automatically when nobody knows about it anymore
- The fact that the DB objects (query object handle, buffer object handle) are called objects is misleading. In fact they should be called widgets
- A must have safety rope for large (OO) systems

GLOBAL SHARED Variables

- Classes and objects don't have access to these
- Static properties are much more powerful anyway
- But to introduce first OO features in an application this may be an issue (when current session management strategy relies on GLOBAL SHARED VARIABLES)

Static classes

- ... classes with only static members
- Easily build a framework that needs no startup
- Usable from procedures and classes
- Build static wrappers to existing framework APIs (like Dynamics managers)
- But can't be unloaded at all! When static classes have access to DB tables, you won't be able to disconnect the DB at runtime

Object-orientation in the ABL

- Introduction
- Comparing OOABL to procedural
- Sample 1: Parameter object
- Sample 2: Static methods, simplified API
- Sample 3: Static events
- Sample 4: FINALLY Block
- Sample 5: CATCH ABL runtime error

Parameter objects

- Simplifies passing of a number of parameter from one routine to another
- Caller and callee may be procedural (procedure or function) or object oriented (method, constructor)
- Allows for optional parameters
- Parameter values may be validated
- Allows for easy extension without changing every caller

Parameter objects

- Supports INPUT and OUTPUT
- Objects are always passed via reference
- Callee can modify properties, caller can query those

Object-orientation in the ABL

- Introduction
- Comparing OOABL to procedural
- Sample 1: Parameter object
- Sample 2: Static methods, simplified API
- Sample 3: Static events
- Sample 4: FINALLY Block
- Sample 5: CATCH ABL runtime error

Static methods, simplified API

- Static methods are invoked on a class, not on an instance
 - Package.ClassName:Method ()
- They are always available (when the R-Code is)
- Like a SESSION SUPER-PROCEDURE
- But no need to instantiate
- Method overloading allows optional and alternative sets of parameters (more than one method with the same name but different parameters)

Static methods, simplified API

- Useful for OO and procedural code
- Allows wrapping of persistent procedures
- Sample: Dynamics manager access

Object-orientation in the ABL

- Introduction
- Comparing OOABL to procedural
- Sample 1: Parameter object
- Sample 2: Static methods, simplified API
- Sample 3: Static events
- Sample 4: FINALLY Block
- Sample 5: CATCH ABL runtime error

Static events

- Static events can be subscribed from everybody
- ... but there is only a single source of that event (no publish from)
- Offers more control, as a static method in the publishing class is required to Publish() the event.
- Allows to check event pre-conditions
- Subscribe from procedural and object oriented code, i.e. Activate event on the AppServer

Object-orientation in the ABL

- Introduction
- Comparing OOABL to procedural
- Sample 1: Parameter object
- Sample 2: Static methods, simplified API
- Sample 3: Static events
- Sample 4: FINALLY Block
- Sample 5: CATCH ABL runtime error

FINALLY Blocks

- A block at the end of an undoable block (including procedures and functions) that will be executed no matter which exit route a code executes
 - completion
 - RETURN statement before the end
 - Runtime error
 - Thrown error (AppError)
- Starting 10.1C this is the place for all cleanup code! (DELETE dynamic object instances etc.)

Object-orientation in the ABL

- Introduction
- Comparing OOABL to procedural
- Sample 1: Parameter object
- Sample 2: Static methods, simplified API
- Sample 3: Static events
- Sample 4: FINALLY Block
- Sample 5: CATCH ABL runtime error

CATCH

- The CATCH Block may be used to handle errors
- Different types of errors
 - ABL runtime error (SysError)
 - Application error (AppError), RETURN ERROR.
 - SOAP Error
 - .NET Exception
- Errors can be caught, handled, ignored and re-thrown

Questions



Thank you

