Consultingwerk
software architecture and development

OOABL for everyday's benefit

Peter Judge, Consultingwerk

Peter Judge

- Senior Architect at Consultingwerk
- Writing 4GL since 1996, working on a variety of frameworks and applications. More recently have worked on a lot of integration-y stuff: Authentication Gateway, HTTP Client, Web Handlers. Dabble in PASOE migrations.
- Active participator in Progress communities, PUGs and other events

Consultingwerk Software Services Ltd.

- Independent IT consulting organization
- Focusing on **OpenEdge** and **related technology**
- Located in Cologne, Germany, subsidiaries in UK, USA and Romania
- Customers in Europe, North America, Australia and South Africa
- Vendor of developer tools and consulting services
- Specialized in GUI for .NET, Angular, OO, Software Architecture, Application Integration
- Experts in OpenEdge Application Modernization

Services Portfolio, Progress Software

- OpenEdge (ABL, Developer Tools, Database, PASOE, …)
- Telerik DevCraft (.NET, Kendo UI, Angular, …), Telerik Reporting
- OpenEdge UltraControls (Infragistics .NET)
- Telerik Sitefinity CMS (incl. integration with OpenEdge applications)
- Kinvey Plattform, NativeScript
- Corticon BRMS
- WhatsUp Gold infrastructure-, network- and application monitoring
- Kemp Loadmaster
- …

Services Portfolio, related products

- Protop Database Monitoring
- Combit List & Label
- Web frameworks, e.g. Angular
- .NET
- Java
- ElasticSearch, Lucene
- Amazon AWS, Azure
- DevOps, Docker, Jenkins, ANT, Gradle, JIRA, …
- …

Agenda

- Intro to OOABL
- OOABL Concepts
- Patterns & practices

The Object-Oriented Extensions to the ABL

- First release was 10.1A in 2005
- Many updates and enhancements over time
  - Structured error handling (10.1C)
  - Abstract classes (10.2B)
  - .NET integration (10.2B)
  - Serialization (11.4)
  - Enum types (11.6)
  - Package protection (12.1)
  - Methods for callbacks (12.3, 12.4)
  - Property overriding (12.5)
  - Generic collections (12.5, 12.6, 12.7)
- And certainly more to come …

Why use OO?

- Memory safety … garbage collector (GC)
- Objects passed by reference
- Compiler helps reduce programming errors through type safety
  - Is the class I'm trying to reference found in PROPATH?
  - Is the method I’m trying to call part of the class?
  - Am I allowed to call the method from here?
  - Can I pass these parameters to the method?

Definitions

Types	A type defines the set of requests to which it can respond; includes classes, interfaces, enums
Strong typing	Compile-time enforcement of rules
Member	Stuff "inside" a type - methods, properties, variables, events etc
Access control	Compile-time restriction on member visibility: public, protected, private, package-*
Class	A type with executable code
Abstract class	A non-instantiable (non-runnable) class that may have executable code
Static	Members that are loaded once per session. think GLOBAL SHARED
Interface	A type with public members without implementations
Enum(eration)	A type with name int64-value pairs
Generic types	A type that gets its type at runtime, rather than when it is written
Object [instance]	Running class

Type names and locations

- Type names contain a package and a "base" name
  - Could have no package but should (and for any production code must)
  - Names have tighter restrictions than procedures and functions
- Packages allow logical groupings of types
  - Can be horizontal layers or vertical slices or something else, up to you
- Package naming
  - First package cannot be Progress … but com.progress can be used
  - Progress also uses OpenEdge for the ABL classes they release
  - We recommend using the top-level / first package for a company/business unit name, similar to a DNS entry

Type names and locations

- A type is represented on disk by a single file with a .cls extension (only)
  - Compiles to a .r
  - The path of the file must be the package name
- For a class named `Consultingwerk.OERA.BusinessEntity`
  - On disk: `c:\SmartComponentLibrary\Consultingwerk\OERA\BusinessEntity.cls`
  - Only `c:\SmartComponentLibrary\` is needed in PROPATH
- Types can be added to PL and APL archives
- Class names are always called with a qualified name (ie path)
  - USING is syntactic sugar to help with long names

Class definition and inheritance

```
CLASS class-type-name [ INHERITS super-type-name]
  [ IMPLEMENTS interface-type-name [ , interface-type-name ] ... ]
  [ USE-WIDGET-POOL ]
  [ ABSTRACT | FINAL ]
  [ SERIALIZABLE ]:
```

- Inheritance allows behaviour to be included – and modified - from a super-class (parent)
- All classes ultimately inherit from Progress.Lang.Object
- Shallow inheritance is better

Access levels

- Restrict which code can access a member
  - Enforced by the compiler
- Defaults vary; be explicit to avoid any confusion
  - Variables = PRIVATE
  - Properties = PUBLIC
  - Methods = PUBLIC
  - Events = PUBLIC
  - Temp-tables = PRIVATE (cannot be PUBLIC)
- It's easier to make the access level less restrictive than more restrictive
  - PROTECTED is a decent default

Access levels (modes)

Access mode	Class	Subclass	Package	All
PRIVATE	✓	✗	✗	✗
PACKAGE-PRIVATE	✓	✗	✓	✗
PROTECTED	✓	✓	✗	✗
PACKAGE-PROTECTED	✓	✓	✓	✗
PUBLIC	✓	✓	✓	✓

https://docs.progress.com/bundle/openedge-oo-abl-guided-journey/page/Access-modes.html

Static members (or, who took my GLOBAL SHAREDs)

- Static members allow access to a class without having to instantiate (aka new or run) a class
  - Includes constructors which run exactly once in a session
- Calling methods, subscribing to events, getting and setting properties all work the same as for objects
- Temp-tables can also be static but that's a whole different topic
- Have many valid uses … but also candidates for abuse
- Aim to use as "helpers" rather than business services

```
catch err as Progress.Lang.Error:
    /* Do something */
    ErrorHelper:ShowErrorMessage(err).
end catch.
```

Defining an API: interfaces and abstract classes

- Both types are a way to define signatures (ie methods and other members) without implementations
  - Guarantees to a caller that a particular method with a particular signature is available in a class
  - Interfaces can only have public members
  - Abstract classes can have public or protected members
- Interfaces allow the definition of smaller APIs that follow the separation of concerns

- Use them!

Collections

- Kinda-sorta like a strongly-typed temp-table for objects
- Different kinds for different use-cases …
  - List : indexed / sorted in numerical order
  - Set : contains unique objects, in no particular order
  - Map / Dictionary : key-value pairs with unique keys

- Navigate through the collection using an iterator or enumerator
- Finding an object in each collection varies

Procedures & Objects

- Procedures can call classes; classes can call procedures
- Pass objects to procedures as parameters
- Allows you to incrementally add OOABL to an application
- Still necessary for certain cases
  - Callbacks (though far fewer now)
  - AppServer event procedures (session start, stop etc)
  - Session start (`-p main.p`)
  - Shared variables
  - "Classic" PUB/SUB

Garbage collector

- Automatically deletes an instance if there are no references to it being held.
  - Same effect as DELETE OBJECT – runs any destructor
- References are held by
  - Variables, Properties, Temp-table fields
  - Event subscriptions
  - Progress.Lang.Object's NEXT-SIBLING and PREV-SIBLING excluded
  - SESSION:FIRST-OBJECT and FIRST-FORM chains excluded
- References are let go by
  - Variables going out of scope
  - ASSIGN <variable | property > = <some other value, including ?>.
  - DELETE OBJECT
  - DELETE temp-table record
- `LOG-MANAGER:LOG-ENTRY-TYPE = 'DynObjects.Class'` shows manual and auto-deletion

Demo

- Value / parameter objects
- Collections, generic and otherwise
- Handle wrappers: procedures, data structures
- Primitive datatype wrappers: memptr, longchar
- Static members
- Use with procedures

Conclusion

- Using OOABL with procedural code is quite easy
  - Most of the OO concepts have similarities to the procedural coding model
  - But the compiler helps much more with OOABL

- OO and procedural ABL can coexist very nicely

PUG Challenge 2024

- Europe: September 18th – 20th in Prague, CZ
- Americas: 29 Sept – 2 Oct, Waltham, MA

Questions


Consultingwerk
software architecture and development