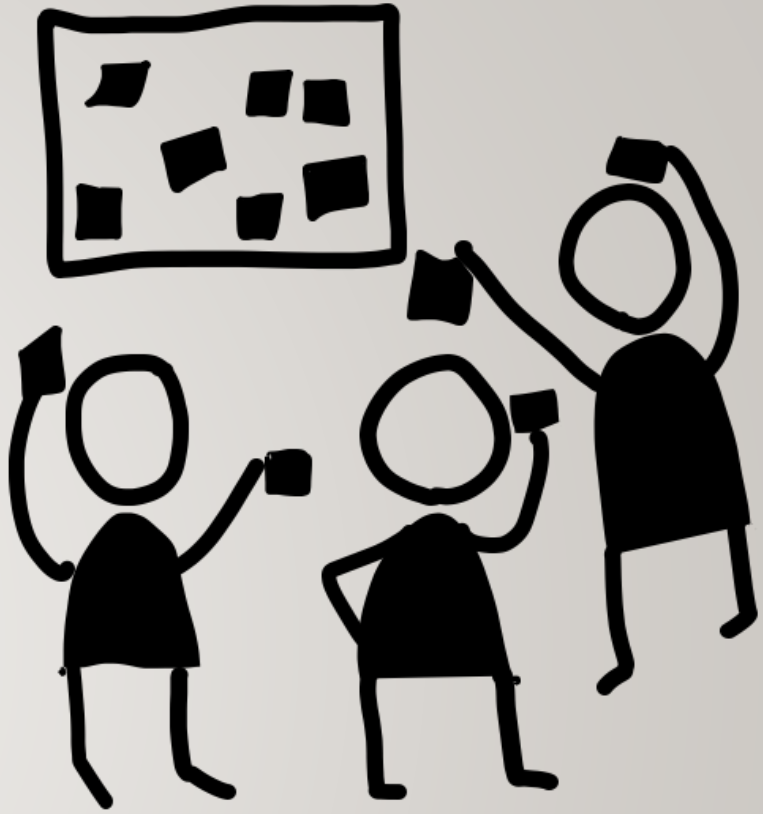
OOABL PATTERNS & PRACTICES

PUG SA Workshop



Mike Fechner **Consultingwerk Ltd** mike.fechner@consultingwerk.de

Peter Judge **Progress Software** pjudge@progress.com

ABSTRACT

- Writing class files is easy. Writing good class files requires a new school of thought – especially for procedural developers. In this session we will discuss about some best practices for the OOABL. We'll be talking about best usage of Enums, Interfaces, Parameter classes and ways to manage dependencies the CCS way. The presenters will be showing OO patterns every developer should know and know when and when not to use them. We'll also be talking about the role of relational concepts like temp-tables and ProDatasets in a class based world.

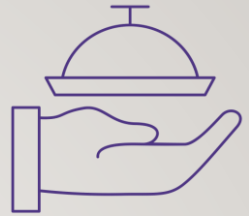
AGENDA – MORNING

- Intro / Welcome
- OO terms and definitions
- Labs context
- Value objects
- Enums
- Strongly-typed Events
- Fluent interfaces
- A Service Manager

AGENDA - AFTERNOON

12.30

- Lunch



13.30

- Composing Complex Apps
  Peter Judge

14.30

- Coffee

14.45

- An introduction into managing dependencies in OOABL applications – or OO patterns every ABL developer should know
  Mike Fechner

DEFINITIONS

Types	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, etc
Access control	compile-time restriction on member visibility: public, protected, private
Class	type with executable code; implementation of a type
Abstract class	non-instantiable (non-runnable) class that may have executable code
Static	members loaded once per session. think GLOBAL SHARED
Interface	type with public members without implementations
Enum(eration)	strongly-typed name int64-value pairs
Object	running classes, aka instance

COMMON TERMS

Software Design Patterns: general reusable solution to a commonly occurring problem within a given context

- Parameter value
- Fluent Interfaces
- Factory method
- Singleton

SOLID principles

S ingle responsibility

Open-closed

L iskov substitution

I nterface segregation

Dependency inversion

http://en.wikipedia.org/wiki/Software_design_pattern

PASTA PATTERNS

SPAGHETTI … A HUGE MESS ALL TANGLED TOGETHER

LASAGNA … TOO MANY UNNEEDED LAYERS ADDING NOTHING

RAVIOLI … JUST ENOUGH LAYERING AND JUST RIGHT ENCAPSULATION

SOFTWARE GOALS

- Flexibility
  - Allow implementations to be swapped without changes to the calling/consuming code
  - Make testing (mocking) easier
- Modularity & extensibility
  - Allow independent (teams / companies) to develop frameworks and components
  - Allow later / third-party extension of components without changing initial / core components

EXTENSIBILITY: PROCEDURES & OBJECTS

- Procedures can call classes; classes can call procedures
- Lets you incrementally add OOABL
- Necessary for certain cases
  - Callbacks
  - AppServer event procedures
  - Session start (`-p main.p`)
- Pass objects to procedures as parameters
- Objects can include includes

LAB: CONNECTION TO YOUR LAB MACHINE

Make sure you can connect to the lab instances on Amazon using some form of Remote Desktop

CONNECTION DETAILS

Wifi details

SSID	**MCC-Conf**
Password	**M1dC0nC3n@cc355**
Voucher	**36320-53575**

Connecting to the VM

User	**\Administrator**
Password	**PUGSA 19!**

SOME USEFUL CONCEPTS

Value objects

enumerations

strongly-typed events & event args

VALUE OBJECTS

> A **value object** is a small object that represents a simple entity whose equality is not based on identity: i.e. two value objects are equal when they have the same value, not necessarily being the same object.
>
> Value objects should be immutable

https://en.wikipedia.org/wiki/Value_object

VALUE OBJECTS

```
/* calling */
oModel:SetValue('Norah').



/* definition */
class Example.Data.Model:
  method public void SetValue (input pValue as character).




end class.
```

VALUE OBJECTS

```
/* calling */
oModel:SetValue('Norah').
oModel:SetValue('Norah', 'x(20)').


/* definition */
class Example.Data.Model:
  method public void SetValue (input pValue as character).
  method public void SetValue (input pValue as character,
                               input pFormat as character).



end class.
```

VALUE OBJECTS

```
/* calling */
oModel:SetValue('Norah').
oModel:SetValue('Norah', 'x(20)').
oModel:SetValue('Norah', 'x(20)', 'UTF-8').

/* definition */
class Example.Data.Model:
  method public void SetValue (input pValue as character).
  method public void SetValue (input pValue as character,
                               input pFormat as character).
  method public void SetValue (input pValue as character,
                               input pFormat as character,
                               input pEncoding as character).
end class.
```

VALUE OBJECTS

```
/* calling */
oString = new Example.String().
oString:Value    = 'Norah'.


oModel:SetValue(oString).

/* definition */
class Example.String:
  define public property Value as character no-undo get. set.


end class.

class Example.Data.Model:
  method public void SetValue (input pValue as Example.String).
end class.
```

VALUE OBJECTS

```
/* calling */
oString = new Example.String().
oString:Value    = 'Norah'.
oString:Format   = 'x(20)'.
oString:Encoding = 'UTF-8'.
oModel:SetValue(oString).

/* definition */
class Example.String:
  define public property Value    as character no-undo get. set.
  define public property Format   as character no-undo get. set.
  define public property Encoding as character no-undo get. set.
end class.

class Example.Data.Model:
  method public void SetValue (input pValue as Example.String).
end class.
```

IMMUTABLE VALUE OBJECTS

```
/* calling */
oString = new Example.String('Norah','x(20)', 'UTF-8').
oModel:SetValue(oString).

// definition
1.  class Example.String:
2.    // read-only properties
3.    define public property Value as character no-undo get. set.
4.    define public property Format as character no-undo get. set.
5.    define public property Encoding as character no-undo get. set.
6.    // values set in constructor(s)
7.    constructor public String(input pValue    as character,
8.                              input pFormat   as character,
9.                              input pEncoding as character):
10.     assign this-object:Value    = pValue
11.            this-object:Format   = pFormat
12.            this-object:Encoding = pEncoding.
13.   end constructor.
14. end class.

class Example.Data.Model:
  method public void SetValue (input pValue as Example.String).
end class.
```

ENUMS

> An **enumerated type** (also called enumeration, enum[…] is a data type consisting of a set of named values called elements, members, enumeral, or enumerators of the type. The enumerator names are usually identifiers that behave as constants in the language
>
> https://en.wikipedia.org/wiki/Enumerated_type

WHY USE ENUMS

```
procedure SetOrderStatus(input pOrderNum as integer,
                         input pStatus as integer):
    OpenEdge.Core.Assert:IsPositive(pStatus, 'Order status').

    find Order where Order.OrderNum eq pOrderNum exclusive-lock.
    assign Order.OrderStatus = pStatus.
    // what happens when the integer isn't a valid status? How do we know ?
end procedure

run SetOrderStatus (12345, 0).   // this throws an error via the Assert
run SetOrderStatus (12345, 1).   // what is this status?
run SetOrderStatus (12345, 2).
```

DEFINING ENUMS

```
// enum type
enum Example.Orders.OrderStatusEnum:  //implicitly FINAL so cannot be extended
  // enum member
  define enum    Shipped          = 1  // default start at 0
                 Backordered           // = 2 . Values incremented in def order
                 Ordered
                 Open
                 Cancelled        = -1 // historical set of bad values
                 UnderReview      = -2
                 Default = Ordered.    // = 3
    // enum members are the only members allowed
    // enum members can only be used in enum types
end enum.
```

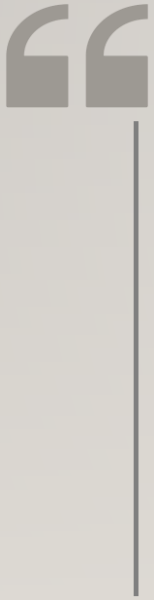
USING ENUMS

```
procedure SetOrderStatus(input pOrderNum as integer,
                         input pStatus   as Example.Orders.OrderStatusEnum):
    //ensures that we have a known, good status
    OpenEdge.Core.Assert:NotNull(pStatus, 'Order status').

    find Order where Order.OrderNum eq pOrderNum exclusive-lock.
    assign Order.OrderStatus = pStatus:GetValue().
    //Alternative way of getting the value
    assign Order.OrderStatus = integer(pStatus).
end procedure

run SetOrderStatus (12345, OrderStatusEnum:None).         // COMPILE ERROR
run SetOrderStatus (12345, OrderStatusEnum:Backordered).
run SetOrderStatus (12345, OrderStatusEnum:Ordered).
```

STRONGLY-TYPED EVENTS & EVENT ARGS



An extensible Publish / Subscribe model with compile-time checking

STRONGLY-TYPED EVENTS: DEFINE

- Name
- Signature
- Access level for subscriptions
- Note: classes cannot subscribe to untyped PUB/SUB events

```
class Example.Data.Model:

  define public event StateChanged signature void (
                  input pData as character).

end class.
```

STRONGLY-TYPED EVENTS: LISTEN

- `Subscribe` & `Unsubscribe`
- Signature-matched handler public method or internal procedure

```
class Example.UI.Grid:
  constructor Grid(input poModel as Example.Data.Model):
     poModel:StateChanged:Subscribe(this-object:StateChangedHandler).
  end constructor.
  method public void StateChangedHandler (input pData as character):
    /* when the data has changed in the model, update the UI */
    this-object:Refresh().
  end method.
  destructor Grid():
     poModel:StateChanged:Unsubscribe(this-object:StateChangedHandler).
 end destructor.
end class.
```

STRONGLY-TYPED EVENTS: RAISE

- Publish is **always** private

```
class Example.Data.Model:
  define public event StateChanged signature void (input pData as char).
  method public void SetValue (input pValue as character):
      /* do stuff with data */
      /* tell the world */
      OnStateChanged(input pValue).
  end method.
  /* Protected On<Event> method lets other classes raise the event */
  method protected void OnStateChanged(input pData as character):
      this-object:StateChanged:Publish(pData).
  end method.
end class.
```

EVENT HANDLERS WITH VALUE OBJECTS

- Event handlers have a standard signature*

```
define public event <name> signature void (input pSender as class Progress.Lang.Object,
                                           input pArgs   as class OpenEdge.Core.EventArgs).
```

  - For static events, sender should be the type sending : `GET-CLASS()`
  - Args are data about the event; operations like `Cancel` too
- A form of value object that follows .NET pattern

  https://blogs.msdn.microsoft.com/kcwalina/2005/11/18/why-do-we-have-eventargs-class/

* Not required by the ABL but should be a best practice

EVENT ARGS EXAMPLE

```
class OpenEdge.Web.DataObject.HandleRequestEventArgs inherits OpenEdge.Core.EventArgs:
  /* (mandatory) The request being serviced */
  define public property Request as IWebRequest no-undo
      get. private set.
  /* (optional) An error that results from the handling of this event. */
  define public property Error as Progress.Lang.Error no-undo
      get. set.
  /* Indicates whether the operation should be cancelled */
  define public property Cancel as logical no-undo
      get. set.
   /* (optional) The status code to return for an operation.
       Zero = use the event args' Response for the entire response */
  define public property ReturnStatusCode as integer no-undo
     get. set.
  /* Constructor.
     @param IWebRequest The request that resulting in the exception */
  constructor public HandleRequestEventArgs(input poRequest as IWebRequest):
    super().
    Assert:NotNull(poRequest, 'Request').
    assign this-object:Request = poRequest.
  end constructor.
end class.
```

FLUENT INTERFACES

> A method for designing object oriented APIs based extensively on method chaining with the goal of making the readability of the source code close to that of ordinary written prose,
>
> https://en.wikipedia.org/wiki/Fluent_interface

Based on Fowler and Evans as described at
https://martinfowler.com/bliki/FluentInterface.html

FLUENT INTERFACES

- Chain calls together without intermediary variables
- Make OO look like a language - make it more readable
- Often used to build a set of parameters (value objects)

```
def var formData as Progress.Lang.Object no-undo.
def var req as OpenEdge.Net.HTTP.IHttpRequest.

req = RequestBuilder:Post('http://httpbin.org/post')
                     :ViaProxy('localhost:8888')
                     :WithData(formData, 'multipart/form-data')
                     :AcceptJson()
          :Request.
```

FLUENT INTERFACES

```
class OpenEdge.Net.HTTP.RequestBuilder abstract:
  /* Returns a usable HTTP Request */
  define abstract public property Request as IHttpRequest no-undo get.

  /* Starts the chain/fluent interface */
  method static public RequestBuilder Post (input pUri as OpenEdge.Net.URI).
  method static public RequestBuilder Build(input pMethod as character,
                                             input pUri as OpenEdge.Net.URI).

  /* 'Language' to modify default behaviour */
  method public RequestBuilder AddHeader(pHeader as OpenEdge.Net.HTTP.HttpHeader).
  method public RequestBuilder AddHeader(input pName as char, input pVal as char).
  method public RequestBuilder ContentType(input pContentType as char).
  method public RequestBuilder AcceptJson ().
  method public RequestBuilder SupportsAuthorization(input plSupport as log).
```

FLUENT INTERFACES: REQUEST BUILDER

```
class OpenEdge.Net.HTTP.DefaultRequestBuilder inherits RequestBuilder:
  define override public property Request as IHttpRequest no-undo
  get():
   oRequest = new OpenEdge.Net.HTTP.HttpRequest().
   if type-of(oRequest, ISupportInitialize) then
     cast(oRequest, ISupportInitialize):Initialize().
   assign oRequest:Method = GetOptionStringValue('method':u)
          oRequest:URI    = cast(GetOptionObjectValue('uri':u), URI).
   for each ConfigOption where ConfigOption.ConfigName begins 'header+':u:
     oRequest:SetHeader(cast(ConfigOption.ObjectValue, HttpHeader)).
   end.
   assign oRequest:Entity = GetOptionObjectValue('entity':u).
   return oRequest.
  end get.
```

LAB: CREATE A FLUENT INTERFACE

Create a fluent interface for the BusinessEntityBuilder class that you have written

SINGLETONS

> The **singleton pattern** is a software design pattern that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system.
>
> https://en.wikipedia.org/wiki/Singleton_pattern

SINGLETONS

TRUE

- Pretty much everyone's first true pattern love
- Use sparingly

FALSE

- Must use STATIC `Instance` PROPERTY
- The only mechanism for "managers"
- Should never be used
- Should always be used

STATIC-INSTANCE-SINGLETON (IF YOU INSIST)

```
1.  class OpenEdge.Example.BusinessLogicManager:
2.      define static public property Instance as BusinessLogicManager no-undo
3.          get():
4.              if not valid-object(BusinessLogicManager:Instance) then
5.                  assign BusinessLogicManager:Instance = new BusinessLogicManager().
6.              return BusinessLogicManager:Instance.
7.          end get.
8.          private set.
9.      // make the default constructor PRIVATE to prevent anyone else NEWing
10.     constructor private BusinessLogicManager():
11.     end constructor.
12.     // normal instance-level method to do work
13.     method public void DoStuff(input pArgument as ValueObject):
14.     end method.
15. end class.
16. // Calling mechanism
17. // static property access (could be stored in a variable)
18. //                        normal object instance operation
19. BusinessLogicManager:Instance:DoStuff(valueObject).
```

WHY NOT?

- Factory + implementation = singleton in a single class
  - Does not allow replacements of any one of those pieces
  - Does not allow mocking (true for static members in general)

- Object contains application (or component) behaviour and lifecycle
  - Not a good separation of concerns

SERVICE MANAGER

> The Service Manager is the infrastructure component that manages the relationship between the abstraction (an OO type name) and the implementing instance (a concrete, instantiable OO type name).

Edu, M., Fechner, M., Prinsloo S. L., Judge, P., & Smith, R. (2016). *Service Manager Specification OpenEdge Application Architecture Specification (OEAA) version 1.0.* Progress Software.

WHAT IS THE COMMON COMPONENT SPECIFICATION (CCS) PROJECT?

- A project for developing standard business application component specifications for business applications, driven by OpenEdge experts and evangelists from the entire Progress Community
- Defines a common understanding and language of an business application architecture
- Enables standards-based framework components that can easily interoperate
- Enables creation of standard tools for those components

CCS OUTPUTS

- Versioned specifications in document form
- Interface definitions – OOABL source code
  - Also includes many interfaces, many enums and one class
  - In 11.7.2 all published interfaces included in OE install

The behavior described in the document and interface(s) MUST be followed in order to claim compatibility with a version of a specification

- May include some sample code or test cases
  - No complete reference implementation as part of the project

https://github.com/progress/CCS

SERVICE MANAGER RESPONSIBILITIES

- The Service Manager provides two categories of functionality
  1. Service name resolution

```
/* Returns a usable instance of the requested service.
   @param  P.L.Class The service name requested
   @return P.L.Object A usable instance
   @throws P.L.AppError Thrown when no implementation can be found */
method public Progress.Lang.Object getService(
                                          input poService as class Progress.Lang.Class).
```

  2. Lifecycle management

```
/* Destroys and flushes from any cache(s) objects scoped to the argument scope.
   @param ILifecycleScope A requested scope for which to stop services. */
method public void stopServices(
                              input poScope as Ccs.ServiceManager.ILifecycleScope).
```

OBJECT LIFECYCLES (AKA "SCOPES")

As long as an object respects a contract – it implements an interface or inherits from an abstract class - does it matter who created it and how?

Or how long it lives for after an object is done using it?

- How long should an object live?
  It could be for the life of the …
  - Session
  - User's login session
  - Request
  - As long as it is used ("transient")
- How many instances should there be?

singleton

COMPLETE CCS.COMMON.ISERVICEMANAGER

```
interface Ccs.Common.IServiceManager inherits Ccs.Common.IManager:
    /* Returns a usable instance of the requested service.
       @param P.L.Class The service name requested
       @return P.L.Object A usable instance */
    method public Progress.Lang.Object getService(input poService as class Progress.Lang.Class).

    /* Returns a usable instance of the requested service.
       @param P.L.Class The service name requested
       @param ILifecycleScope A requested scope. The implementation may choose to ignore this value.
       @return P.L.Object A usable instance */
    method public Progress.Lang.Object getService(input poService as class Progress.Lang.Class,
                                                  input poScope as Ccs.ServiceManager.ILifecycleScope).

    /* Returns a usable instance of the requested service.
       @param P.L.Class The service name requested
       @param character An alias for the service. The implementation may choose to ignore this value.
       @return P.L.Object A usable instance */
    method public Progress.Lang.Object getService(input poService as class Progress.Lang.Class,
                                                  input pcAlias as character).

    /* Destroys and flushes from any cache(s) objects scoped to the argument scope.
       @param ILifecycleScope A requested scope for which to stop services. */
    method public void stopServices(input poScope as Ccs.ServiceManager.ILifecycleScope).
end interface.
```

USING A SERVICE MANAGER

- How does it know what implementation to return for a requested interface (type)?
  - Use a database- or temp-tables as some form of registry
  - Pattern matching
- Load mappings from
  - JSON or XML
  - Code
- How do you get a reference to it?

CCS.COMMON.APPLICATION CLASS

- The only class CCS will ever publish
- Naming it *Framework* seemed wrong
- Provides access to the Startup Manager
- Provides access to the Service Manager (for convenience)
- Yes – it is like a GLOBAL SHARED variable

  … but there didn't seem to be a better way

CCS.COMMON.APPLICATION

```
CLASS Ccs.Common.Application FINAL:
  // Provides access to the injected IStartupManager.
  DEFINE STATIC PUBLIC PROPERTY StartupManager AS Ccs.Common.IStartupManager NO-UNDO GET. SET.

  // Provides access to the injected IServiceManager.
  DEFINE STATIC PUBLIC PROPERTY ServiceManager AS Ccs.Common.IServiceManager NO-UNDO GET. SET.

  //  Provides access to the injected ISessionManager.
  DEFINE STATIC PUBLIC PROPERTY SessionManager AS Ccs.Common.ISessionManager NO-UNDO GET. SET.

  // Version of the Common Component Specification implementation.
  DEFINE STATIC PUBLIC PROPERTY Version AS CHARACTER NO-UNDO INITIAL '1.0.0':u
    GET.
  // Prevent creation of instances.
  CONSTRUCTOR PRIVATE Application ():
    SUPER ().
  END CONSTRUCTOR.
END CLASS.
```

USING CCS.COMMON.APPLICATION

```
/* Invokes/instantiates a webhandler */
method private IWebHandler InvokeHandler(input pHandlerName as character):
  define variable webHandler as IWebHandler no-undo.

  assign webHandler = cast(Ccs.Common.Application:ServiceManager
                           :getService(get-class(Progress.Web.IWebHandler),
                                       pHandlerName           ),
                          IWebHandler) no-error.
  if not valid-object(webHandler) then
  do:
    webHandler = dynamic-new pHandlerName ().
    if type-of(webHandler, ISupportInitialize) then
      cast(webHandler, ISupportInitialize):Initialize().
  end.
  return webHandler.
end method.
```

LAB: IMPLEMENT A SERVICE MANAGER

Use the CCS IServiceManager interface

Implementation can call the builders already created

Update the service interface calling code to use this implementation

WHAT DID WE LEARN?

- OOABL terminology
- Using (immutable) value objects event handlers
- Enumerations
- How to create and use fluent interfaces

USEFUL LINKS / SITES

https://github.com/PeterJudge-PSC/Composing_Complex_Apps

https://github.com/progress/CCS

https://github.com/consultingwerk/CCS_Samples

THINGS FOR LATER & ANOTHER TIME

- Interfaces & abstract classes
- Factories & the abstract builder
- Dependency injection & Service Managers
- Domain-Driven Design
- Working with relational data
- Generic programming using reflection
- Garbage collection
- Facades / Decorators & Adapters
- "GUI" for .NET

Peter Judge pjudge@progress.com
Mike Fechner mike.fechner@consultingwerk.de