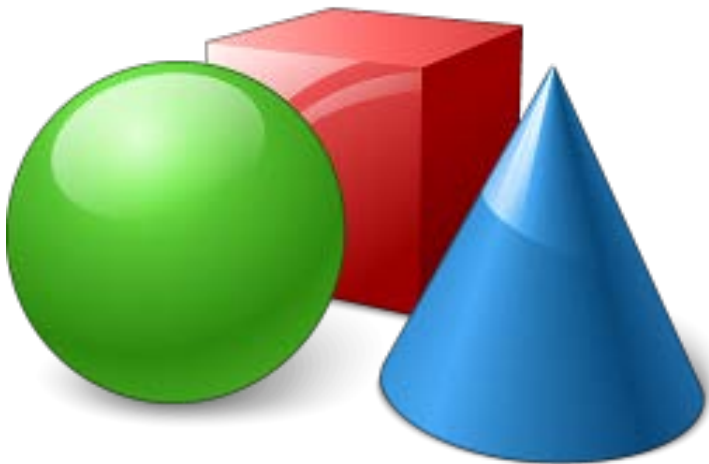


OO-Oh!



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



Consultingwerk Ltd.



- Independent IT consulting organization
- Focusing on **OpenEdge** and **related technology**
- Located in Cologne, Germany
- Customers in Europe, North America, Australia and South Africa
- Vendor of tools and consulting programs
- 26 years of Progress experience (V5 ... OE11)
- Specialized in GUI for .NET, OO, Software Architecture, Application Integration

Audience questions

- OO-ABL coders anybody?
- Who has ever implemented an Interface?
- Who has ever defined an Interface?
- Who has ever written an Abstract class?
- Who has ever inherited from a Class?
- Who has ever defined custom Error classes?
- Who knows the ABL reflection API by heart?

CLASS Statement

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

- Only a single CLASS statement per .cls file
- No nested types
- A class forms a data-type in ABL runtime

CLASS Statement

- Class Type Name: Class Name including relative path (package)
- INHERITS: Base class name (full or based on USING)
- IMPLEMENTS: 0, 1 or multiple Interface types implemented by the class
- USE-WIDGET-POOL: Create an unnamed widget pool scoped to class instances or the static portion
- SERIALIZABLE: Allow serialization across AppServer boundary (AppError derived)

CLASS Statement

- **ABSTRACT**
 - Class cannot be instantiated
 - Requires child class to inherit from
 - May have abstract members
 - Class can only be used as “include file”
- **FINAL**
 - Class cannot be inherited from
- **ABSTRACT and FINAL are mutually exclusive**

OOABL: Members of a class

- Constructor(s)
- Destructor
- Methods, overloaded methods, polymorphic methods
- Data Members
 - Properties
 - Variables (primitive and reference types)
 - Defined non OO objects, Temp-Tables, ProDatasets, Query, Buffer, ...
- Events
- Static or instance based

PRIVATE or PUBLIC or PROTECTED

- Properties and Methods (including Constructor) can be PRIVATE/PROTECTED/PUBLIC
- **PUBLIC is generally a bad default**
- Can't change signature of a PUBLIC method without potentially harming consumers
- PUBLIC should only be chosen when there is a requirement for another class to call into that method, access that property

PRIVATE or PUBLIC or PROTECTED

- PRIVATE/PROTECTED is not about hiding the implementation. It's about avoiding the risk of **having to support** those API's
- Our default is PROTECTED. Less strict than PRIVATE, our framework designed to support customization also through inheritance
- We use PRIVATE only where we have good reasons, because we want to remain able to change without notice
- Others may have PRIVATE as default

Inheritance



CLASS ChildClass INHERITS BaseClass

Inheritance



CLASS GrandChildClass INHERITS ChildClass

CLASS ChildClass INHERITS BaseClass

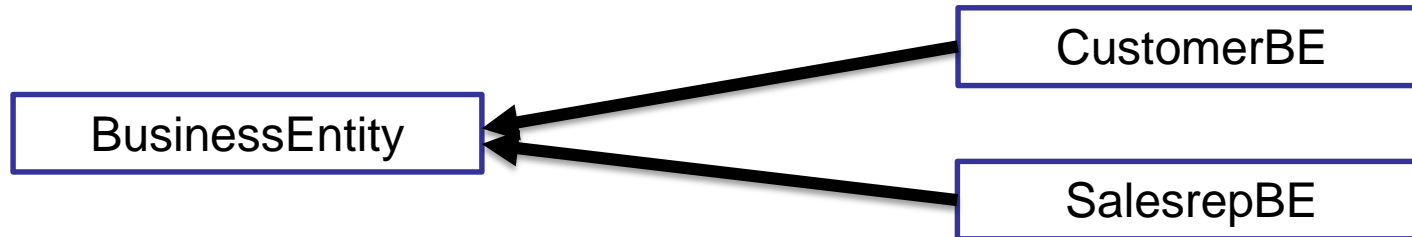
Inheritance

- The *Include* file in the OO world
- Child Class inherits behavior of Base Class
 - Similar to *including* Properties, Methods and Events
- Methods of Base class accessible on instance of the Child class, ability to “OVERRIDE”
- No override for properties
- Constructors are not inherited
 - Relevant as soon as constructors require parameters to call them

Inheritance

- Parent Class can be used on it's own unless
 - it has only a PRIVATE/PROTECTED Constructor
 - is defined as ABSTRACT
- ABSTRACT is not the proper indication that a class cannot be instantiated. ABSTRACT declares an Implementation requirement
- PRIVATE/PROTECTED Constructor disallows instance creation (static helper class)

Inheritance



CLASS CustomerBE INHERITS BusinessEntity
CLASS SalesrepBE INHERITS BusinessEntity

Inheritance

- Inheritance creates multiple classes (aka Types) that can be referenced with Variables of that type

```
/* ***** Definitions *****  
  
USING Consultingwerk.SmartComponentsDemo.OERA.Sports2000.* .  
  
DEFINE VARIABLE oEntity AS Consultingwerk.OERA.BusinessEntity NO-UNDO .  
  
/* ***** Main Block *****  
  
oEntity = NEW CustomerBusinessEntity () .  
  
// -----  
  
oEntity = NEW SalesRepBusinessEntity () .
```

Inheritance

- Consumer of Child Classes of the Base Class knows that all Child Classes provide at the minimum the METHOD's, PROPERTY's, EVENT's of the Base Class
- There may be additional ones

Abstract Base Classes

- If Base Class is incomplete or not usable on it's own, it can be defined as ABSTRACT
 1. Can't new Base Class
 2. Can have ABSTRACT METHOD's
 3. Can have ABSTRACT PROPERTY's
 - ABSTRACT PROPERTY's can be overridden in Child class

Abstract Base Classes

- Protects developers from assuming they can use the Base class
- ABSTRACT METHOD's allow the Base Class to call in a Child Classes METHOD without DYNAMIC-INVOKE or any CAST
 - ABSTRACT METHOD's can be PUBLIC, PROTECTED and PRIVATE
- Inner-class callback

```
/*-----  
Purpose: Saves changes using the DataAccess object  
Notes: This method saves changes contained in the dataset currently  
in the Business Entity  
-----
```

```
METHOD PUBLIC VOID SaveChanges ():
```

```
THIS-OBJECT RequestType = ConsultingRequestType.FromRequestTypeFrom
```

```
THIS-OBJECT:ValidateData() .
```

```
IF NOT THIS-OBJECT DataSetHandler THEN END
```

```
IF NOT VALID-OBJECT (THIS-OBJECT DataSetHandler) THEN  
THIS-OBJECT InitializeDataSetHandler () .
```

```
/*-----  
Purpose: Provides a hook for high level data  
operations  
Notes: Invoked during SaveChanges (). When  
is set, the Update operation will  
the data to the database using the  
Abstract method to be implemented  
-----
```

```
METHOD PUBLIC ABSTRACT VOID ValidateData ().
```

```
END.
```

Abstract Base Classes

- Implementation detail between Child Class and Base Class
- Consumer of the Child Class should not care about this minor detail

Interfaces

- Interfaces enforce implementation of METHOD's, PROPERTY's, EVENT's without providing an implementation
- Interfaces provide a common type like Base Classes do
- Interfaces guarantee consumer of a Class certain methods
- Interfaces do not limit “creativity” during implementation as requiring to inherit would

Interfaces

- Do only contain PUBLIC members
 - PROTECTED and PRIVATE members are an implementation detail
- Do not contain CONSTRUCTORS
 - Factory classes still need to know this implementation detail
 - i.o.W. there is no way to enforce availability of certain constructors

Interfaces

```

/* ***** Definitions *****
USING Consultingwerk.SmartComponentsDemo.OERA.Sports2000.* .
DEFINE VARIABLE oEntity AS Consultingwerk.OERA.IBusinessEntity NO-UNDO .


/* ***** Main Block *****

oEntity = NEW CustomerBusinessEntity () .

// -----

oEntity = NEW SalesRepBusinessEntity () .

```



Interfaces

- When instances of classes are passed as parameters or returned from methods, INTERFACE's are preferable over Base Classes or Child Classes as parameters
 - There are hardly any cases where a class is the best solution here
- Specification on requirements only, not enforcing implementation details (like Base class)

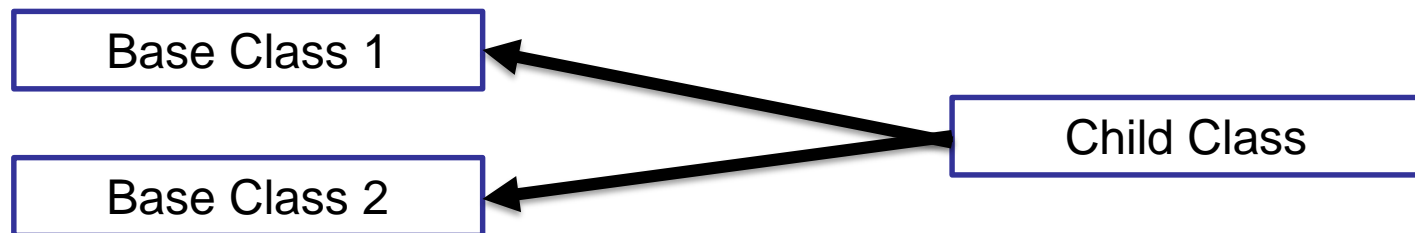
Interfaces Sample

- CCSBE IBusinessEntity
 - IUpdatable BusinessEntity
 - GetDataRequest etc.



Multiple-Inheritance

- Mawg – “half man, half dog”



Multiple-Inheritance

- Like most OO Languages ABL does not support multiple inheritance
- ABL does support multiple Interface implementation
 - So based on Interfaces “Barfolomew” can be of type IMan and Idog
 - Google for “composition over inheritance” or “delegation over inheritance”


```
CLASS Spaceballs.Barfolomew IMPLEMENTS IDog, IMan:
```

```
DEFINE VARIABLE oMan AS ManKind NO-UNDO .  
DEFINE VARIABLE oDog AS DogKind NO-UNDO .
```

```
CONSTRUCTOR PUBLIC Barfolomew ():  
    SUPER ().
```

```
    oMan = NEW ManKind () .  
    oDog = NEW DogKind () .
```

```
END CONSTRUCTOR.
```

```
METHOD PUBLIC VOID Bark (pcWords AS CHARACTER):
```

```
    oDog:Bark (pcWords) .
```

```
END METHOD .
```

```
METHOD PUBLIC VOID Speak (pcWords AS CHARACTER):
```

```
    oMan:Speak (pcWords) .
```

```
END METHOD .
```

```
END CLASS.
```

type-concept via Interfaces

Implementation via
Delegate classes

CAST

- Compiler attempts to verify type compatibility
 - assignment of reference variables
 - passing object references as parameters
- With CAST developer says, “I know better than compiler”
- Type checking delayed from compile time to runtime
- CAST from more generic type reference to a more specific one

CAST with CCSBE Service Manager

- Demo

```
interface CCS.Common.IServiceManager inherits IManager:  
  
    method public IService getService( input poServiceClass as Progress.Lang
```

```
DEFINE VARIABLE oMatchesService AS IEuro2016ResultService NO-UNDO .
```

```
oMatchesService = CAST (Application:ServiceManager:getService  
                        (GET-CLASS (IEuro2016ResultService)),  
                        IEuro2016ResultService) .
```

```
oResults = oMatchesService:GetTodaysMatchResults().
```

CAST with UltraToolbarsManager

- UltraToolbarsManager has a list of „ToolBase“ instances
- ToolBase common set of properties for all tools on the Ribbon, like Enabled, Text
- StateButtonTool (Checkbox) has additional properties, like Checked
- When retrieving a StateButtonTool reference from the Tools collection, you must CAST to be able to access the Checked property

CAST with UltraToolbarsManager

```

/*-----*/
Purpose: Returns the Checked property of a StateButtonTool
Notes:
@param poToolbarsManager The reference to the UltraToolbarsManager that contains the StateButton Tool
@param pcToolKey The key of the StateButton Tool in the UltraToolbarsManager Tools Collection
@return Logical value indicating of the StateButtonTool is checked or not
-----*/
METHOD PUBLIC STATIC LOGICAL GetStateButtonToolChecked (poToolbarsManager AS UltraToolbarsManager,
                                                         pcToolKey AS CHARACTER):

    DEFINE VARIABLE oStateButtonTool AS StateButtonTool NO-UNDO .

    IF NOT poToolbarsManager:Tools:Exists (pcToolKey) THEN
        UNDO, THROW NEW AppError (SUBSTITUTE ("Invalid ToolKey: &1"#{&TRAN}, pcToolKey), 0) .

    IF NOT TYPE-OF (poToolbarsManager:Tools[pcToolKey],
                   StateButtonTool) THEN
        UNDO, THROW NEW AppError (SUBSTITUTE ("Tool &1 is not a StateButtonTool."#{&TRAN}, pcToolKey), 0) .

    oStateButtonTool = CAST (poToolbarsManager:Tools[pcToolKey],
                             StateButtonTool) .

    RETURN oStateButtonTool:Checked .

END METHOD.

```

CAST (CAST (o, P.L.Object), Type)

- There are rare scenarios where a single CAST is not enough
- CASTING from an interface to a base class that does not directly implement that interface
 - typically when working with a base class that you can't influence
 - GUI for .NET
- Compiler will not accept CAST as Base class and Interface are incompatible
- Fool the Compiler with CAST(CAST())

```

/*-----
Purpose: Adds an additional ISmartTabFolderPage based Control to an existing
        Tabfolder Page
Notes:   The tab page with the given key must already exist
@param poTabFolderPage The reference to the ISmartTabFolderPage to add to the tab page
@param pcPageKey       The key of the tab folder (see CreateTabPage) to add the control to
@param pcTabKey        The key for the ISmartTabFolder Page
@param poDataSource    The reference to the ISmartDataSource to pass through to the ISmartTabFolder
@param poTableIOSource The reference to the ISmartTableIOSource to pass through to the ISmartTabF
@param poNavigationSource The reference to the ISmartNavigationSource to pass through to the ISma
@return The reference to the added ISmartTabFolderPage (fluid coding style)
-----*/

```

```

METHOD PROTECTED ISmartTabFolderPage AddToTabPage (poTabFolderPage AS ISmartTabFolderPage,
                                                    pcPageKey AS CHARACTER,
                                                    pcTabKey AS CHARACTER,
                                                    poDataSource AS ISmartDataSource,
                                                    poTableIOSource AS ISmartTableIOSource,
                                                    poNavigationSource AS ISmartNavigationSource):

    DEFINE VARIABLE oControl AS System.Windows.Forms.Control NO-UNDO .
    DEFINE VARIABLE oTab AS UltraTab NO-UNDO .

    IF NOT THIS-OBJECT:ultraTabControl1:Tabs:Exists (pcPageKey) THEN
        UNDO, THROW NEW AppError ("Invalid tab page key.", 0) .

    oTab = THIS-OBJECT:ultraTabControl1:Tabs [pcPageKey] .

    oControl = NEW UltraSplitter () .
    oControl:Dock = DockStyle:Bottom .

    oTab:TabPage:Controls:Add (oControl) .

    oControl = CAST (CAST (poTabFolderPage, Progress.Lang.Object), System.Windows.Forms.Control) .
    oControl:Dock = DockStyle:Bottom .

```


TYPE-OF

- Verifies if CAST is possible
- Returns true, when type implements Interface
- Returns true, when type is child type
- Returns true, when types are the same

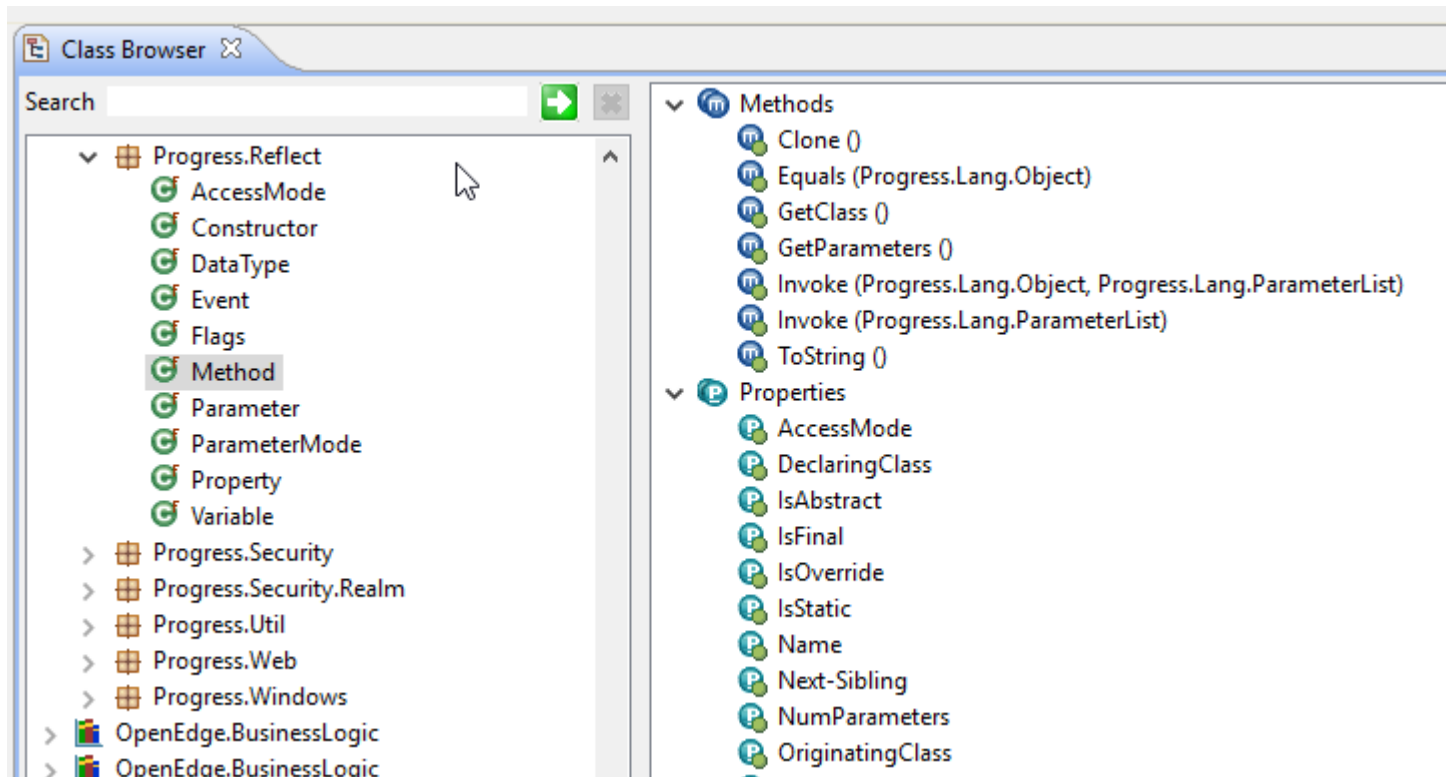
Reflection

- Dynamic access to methods and properties
 - DYNAMIC-NEW
 - DYNAMIC-INVOKE
 - DYNAMIC-PROPERTY
-
- Similar to RUN VALUE(), DYNAMIC-FUNCTION, hProcedure:INTERNAL-ENTRIES

Reflection API in 11.6

- Allows to query a class for properties, methods and constructors at runtime
- Allows for fully dynamic (configurable) INVOKE-METHOD
- Based on classes in Progress.Reflect package

Reflection API in 11.6



Reflection API in 11.6

```

/*-----
Purpose: Returns a comma delimited list of all method names with the given
        number of parameters and flags
Notes:   Used by the DataAccess class to retrieve the names of validation methods
@param poClass The class to return the method names from
@param piNumParameters The number of parameters to return
@param poFlags The method flags to filter methods
@return The comma delimited list of method names with the given number of parameters
-----*/
METHOD PUBLIC STATIC CHARACTER GetMethodNamesByParameterCount (poClass AS Progress.Lang.Class,
                                                                piNumParameters AS INTEGER,
                                                                poFlags AS Progress.Reflect.Flags):

    DEFINE VARIABLE cMethodNames AS CHARACTER NO-UNDO.

    DEFINE VARIABLE oMethods AS Progress.Reflect.Method NO-UNDO EXTENT .

    DEFINE VARIABLE i AS INTEGER NO-UNDO.

    {Consultingwerk/Assertion/ObjectAssert/IsValid.i poClass "'poClass':U"}
    Assert:GE (piNumParameters, 0) .

    oMethods = poClass:GetMethods (poFlags) .

    DO i = 1 TO EXTENT (oMethods):

        IF oMethods[i]:NumParameters = piNumParameters THEN
            cMethodNames = ListHelper:AddEntry (cMethodNames, oMethods[i]:Name) .

    END.

```

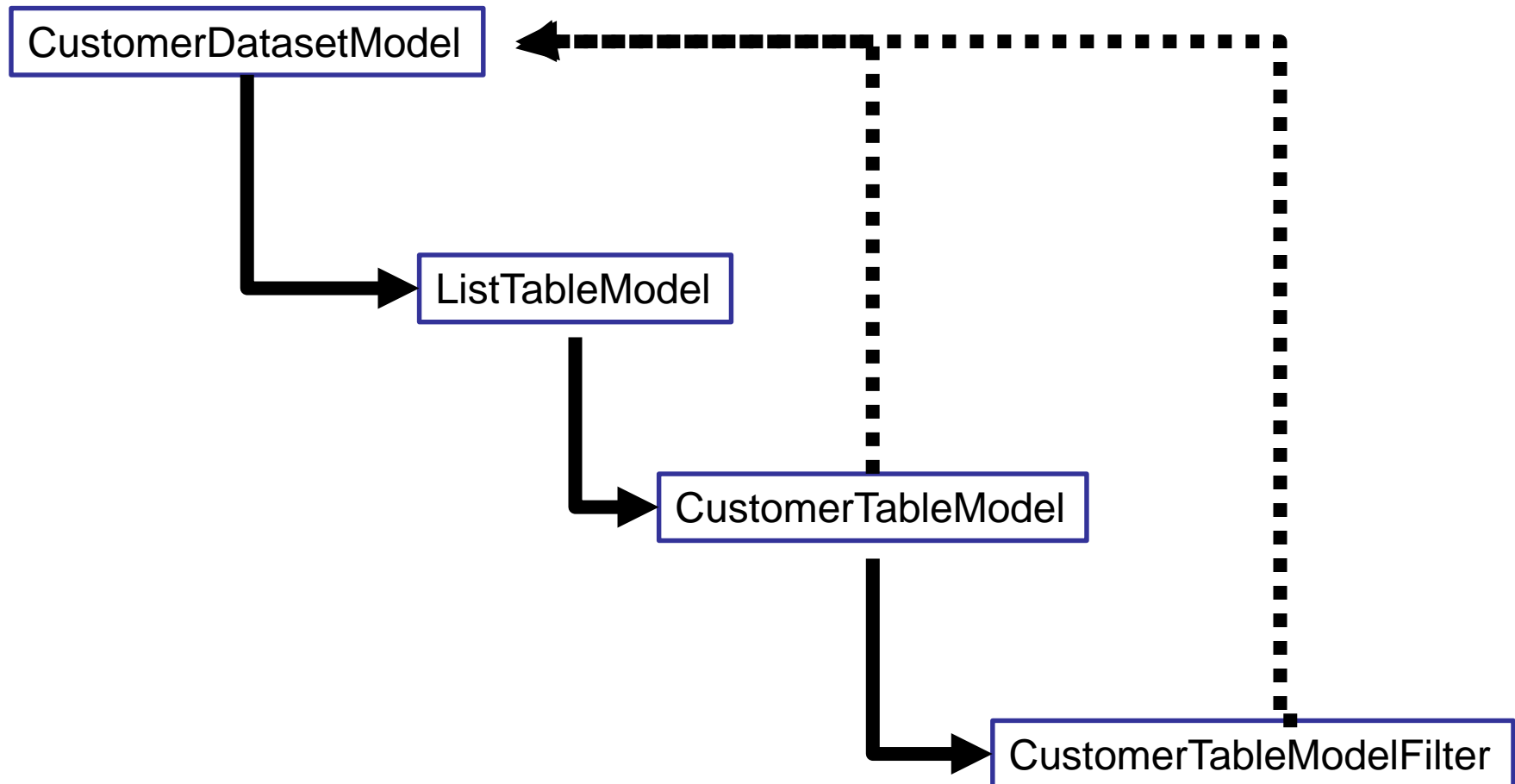
Reflection

- Don't use reflection, just because you can
- Great way of giving up many OO benefits ...
 - Interfaces and CAST are safer way
- Reflection is typically used in central framework components
 - OERA Service Interface
 - Developer tools
- Reflection should be avoided in regular business logic

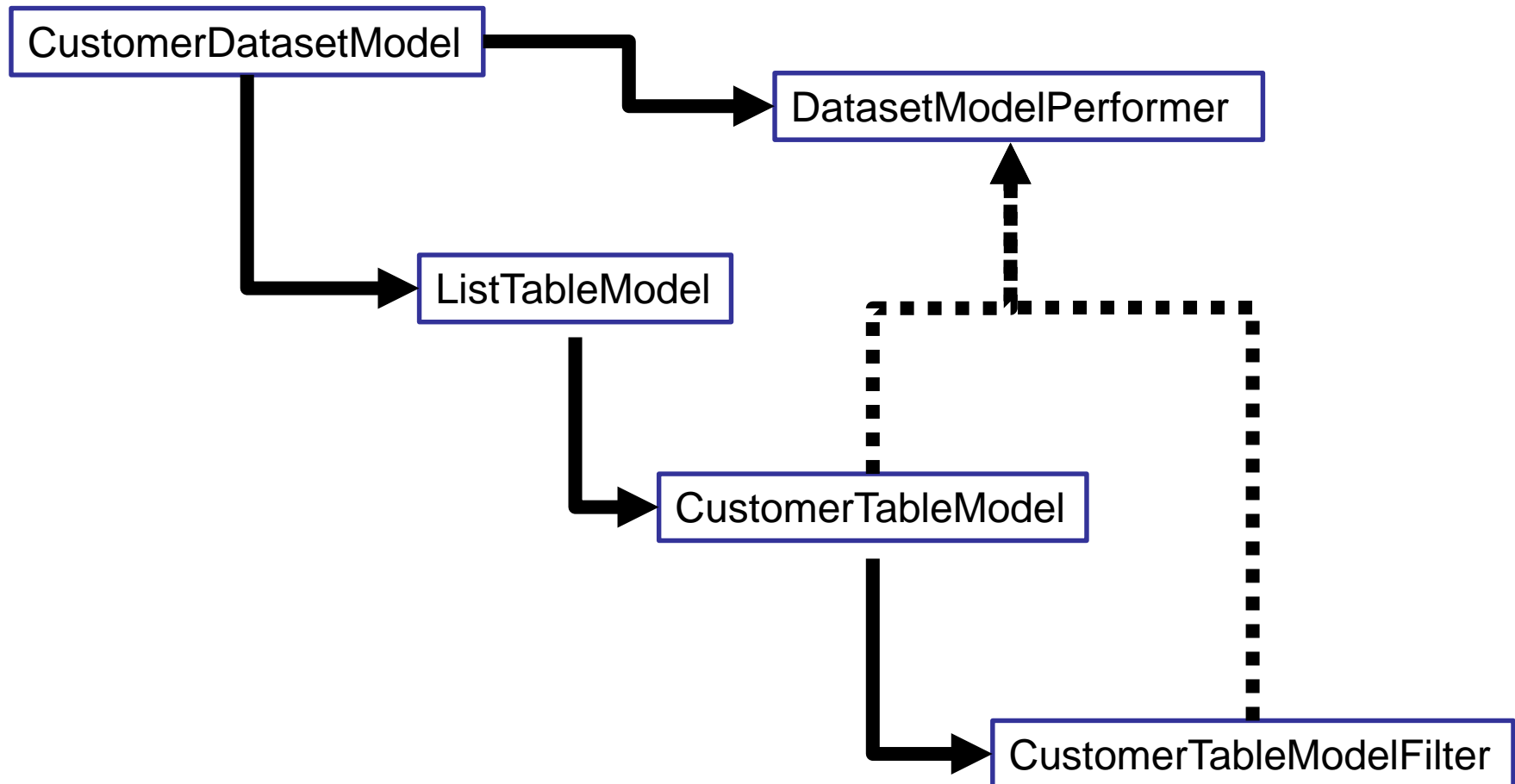
DatasetModel in SmartComponent Library

- Demo ...
- Create new Business Entity with Dataset Model
- Review generated code

DatasetModel Problem



Solution: Split DatasetModel class into 2



Solution: Split DatasetModel class into 2

- By moving those parts of Class A that class B and C depend on into a separate class, the circular reference could be resolved
- DatasetModel construct does no longer hold a circular reference by itself
- As soon as no consumer does no longer hold a reference to the DatasetModel instance, it's GC'd from memory
- No need for DELETE OBJECT

Events Publishing the .NET way

- Events are typically defined PUBLIC
 - That controls the ability to SUBSCRIBE
- Events can only be published from the defining class
 - :PUBLISH is always PRIVATE
- Event subscribers receive event callbacks in FIFO order
- Similar implementation in .NET

On...EventName method

- ▼ Constructors
 - Button ()
- > Methods
- > Properties
- ▼ Events
 - AutoSizeChanged
 - BackColorChanged
 - BackgroundImageChanged
 - BackgroundImageLayoutC
 - BindingContextChanged
 - CausesValidationChanged
 - ChangeUICues
 - Click
 - ClientSizeChanged
 - ContextMenuChanged

Button.OnClick Method (EventArgs)

.NET Framework 4.6 and 4.5 | [Other Versions](#) ▼

Raises the [Click](#) event.

Namespace: [System.Windows.Forms](#)

Assembly: [System.Windows.Forms \(in System.Windows.Forms.dll\)](#)

Syntax

C#	C++	F#	VB
<pre>protected override void OnClick(EventArgs e)</pre>			

Parameters

e

Type: [System.EventArgs](#)

An EventArgs that contains the event data.

On...EventName Method

- Only place to PUBLISH the Event
- PROTECTED METHOD
 - Callable from Child Class
 - Overridable in Child Class
- In the Base Class that defines the event the On...EventName Method allows to verify event pre-conditions before PUBLISH
- In the Child Class an override to the On...EventName method allows to handle event before OR after any body else

Demo

- Progress Developer Studio for OpenEdge macro to define events: “defevent”

```
/*-----  
    Purpose: Raised ${cursor}  
    Notes:  
    @param sender The object that raised the ${event} event  
    @param e The ${eventargs} with the data for the event  
-----*/  
DEFINE PUBLIC EVENT ${event} SIGNATURE VOID (sender AS Progress.Lang.Object,  
                                             e AS ${eventargs}).  
  
/*-----  
    Purpose: Raises the ${event}  
    Notes:  
    @param e The ${eventargs} with the data for the event  
-----*/  
METHOD PROTECTED VOID On${event} (e AS ${eventargs}):  
  
/*    Consultingwerk.Assertion.EventArgsAssert:IsValid (e, "${event}":U) . */  
  
    IF NOT VALID-OBJECT (e) THEN  
        e = Consultingwerk.EventArgs:Empty .  
  
    THIS-OBJECT:${event}:Publish (THIS-OBJECT, e) .  
  
END METHOD .
```


Class Naming Recommendations

- **Keep class names unique over all packages!!!**
- Makes copy & paste programming less error prone
- Don't do:
 - Consultingwerk.Frontend.Views.Customer
 - Consultingwerk.Data.Models.Customer
 - Consultingwerk.Backend.Customer
- Rather do: CustomerView, CustomerModel, CustomerBusinessEntity

PDSOE on Steroids ...

- <http://oedt.hh-berlin.de>

Oedt Editor x

← → ↻ <https://www.hh-berlin.de/oedt/features/editor>

Oedt Plugin Home Features Installation Preferences FAQ

50 END METHOD.

Method Argument Completion

Maybe the most useful code assist feature. You use a method in the content assistant and Oedt insert the parameter names and start a content assistant for the datatypes that are valid.

```
/**
 *
 * @param oBranch : input hhberlin.buildsystem.git.GitBranch
 */
METHOD VOID createFileList(oBranch AS GitBranch):
    fBuildLog:logMessage("Create file list").
    DEFINE VARIABLE cTemp Message loggen
    cTemp = 'cmd /c ~"Pus @param cMessage: input character + "src\ & dir /A:-D-H-S /s /b > "
```

fBuildLog:logMessage(cMessage)

OS-COMMAND SILENT VAL

END METHOD.

Message loggen

@param cMessage: input character

cTemp: character

ToString(): character

I cTemp: character

hhberlin.buildsystem.common.FileBusinessEntity

Don't miss our other presentations

- Monday 11:00: **CCS Deep Dive** (Mike)
- Tuesday 11:00: **OO-Oh** (Mike)
- Tuesday 13:00: **Application Modernization using the SmartComponent Library** (Mike and Marko)
- Tuesday 16:45: **REST in Peace** (Mike)
- Wednesday 11:00: **CCS BoF** (all CCS)
- Wednesday 11:00: **Angular JS for OpenEdge programmers** (Marko)



Questions

