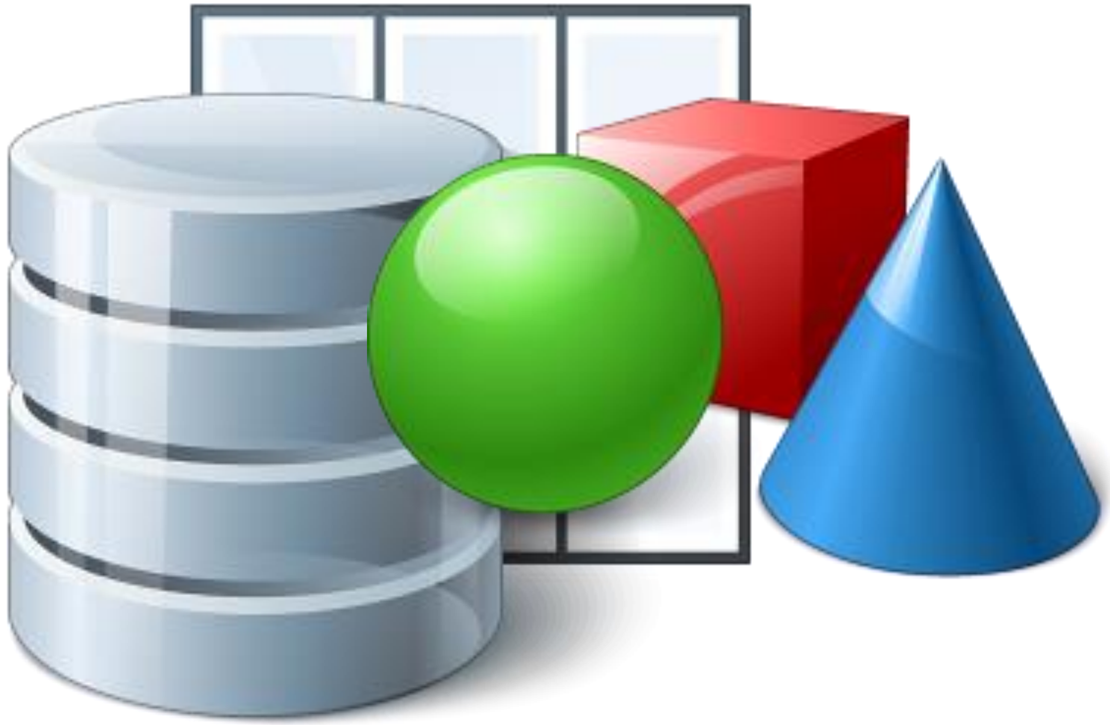
Consultingwerk
software architecture and development

Managing Data in an Object World



Mike Fechner, Director, Consultingwerk Ltd.

mike.fechner@consultingwerk.de

Consultingwerk Ltd.

- Independent IT consulting organization
- Focusing on **OpenEdge** and **.NET**
- Located in Cologne, Germany
- Vendor of tools and consulting programs
- 24 years of Progress experience (V5 … OE11)
- Specialized in GUI for .NET, OO, Software Architecture Application Integration

OpenEdge® application modernization solutions

- WinKit
- **SmartComponent Library**
- Dynamics4.NET
- Tools can be used together or separately
- Share common code base
- SmartComponents.Mobile, SmartComponents.Web
- SmartBPMAdapter for OpenEdge BPM/Savvion
- Smart Rollbase Adapter for OpenEdge

Agenda

- Rules Showcase
- Value Objects
- Object Relational Mapping
- Business Entities
- Dataset Model Classes
- Conclusion

Rules Showcase

- This talk will give you an overview of different approaches for dealing with data in an OOABL application
- All are right – all are wrong …
- But they have different value strength for different use cases
- So, you will need to pick yours…

Rules Showcase

- Object Oriented principles
- Suited for the ABL, ABL suited for model
- AppServer support
- User Interface support

Object Oriented principles

- Encapsulation
- Extensibility
- Code Reuse
- Single Responsibility Principle

Suited for the ABL, ABL suited for model

- Implementation effort
- Queryablility
- Transaction support, before image management
- Ad hoc use

AppServer support

- Effort needed to use model on the AppServer
- Support for AppServer clients
- ABL
- .NET
- Java
- Web Services
- REST (including OpenEdge Mobile and Rollbase)

User Interface Support

- Ability to bind UI to data
  - Viewer
  - Browser / Grids
  - Navigation

Agenda

- Rules Showcase
- Value Objects
- Object Relational Mapping
- Business Entities
- Dataset Model Classes
- Conclusion

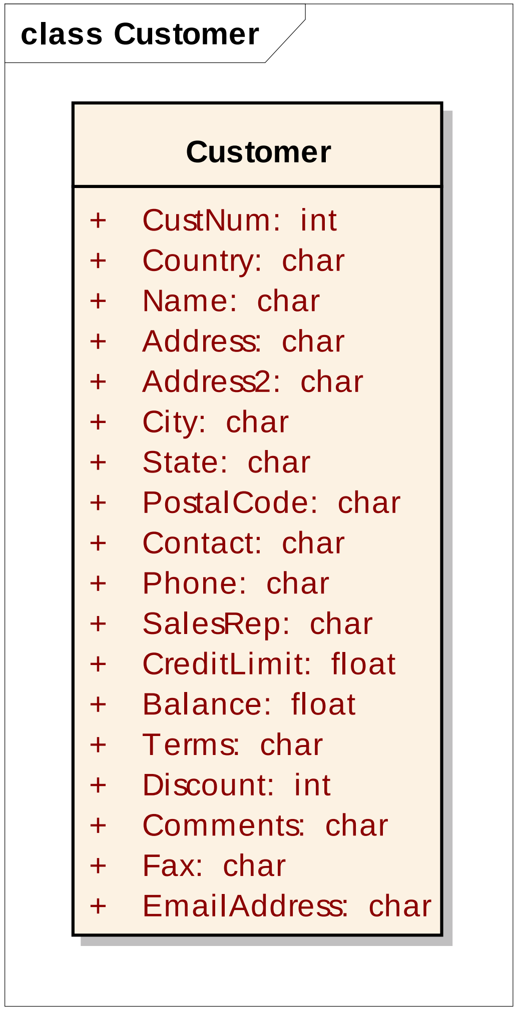
Value Objects

- Simple objects with mostly properties
- Perfect for passing information around
- Can be used as parameter objects (currently within a session only, 11:4 ABL client to ABL AppServer)
- Similar to struct’s in C-like languages
- Easy use with procedural code

Value Objects

- Tim Kuehn’s “single record temp-tables”
- Advantage over temp-tables with small amount of records: does not blows up the DBI file
- Beauty in code: temp-tables and ProDatasets are defined per compile unit, objects may be defined inside internal procedures, methods, more local scope
- Life time (object + data) may end as soon as a method is left
- Temp-Table (definition + data) stays longer

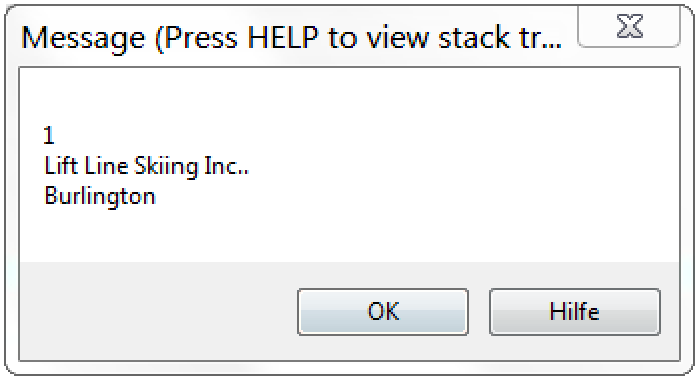
Value Objects



```
CLASS Demo.ManagingDataInObjects.SimpleValueObject.Customer:

    DEFINE PUBLIC PROPERTY CustNum AS INTEGER  NO-UNDO
    GET.
    SET.

    DEFINE PUBLIC PROPERTY Country AS CHARACTER  NO-UNDO
    GET.
    SET.

    DEFINE PUBLIC PROPERTY Name AS CHARACTER  NO-UNDO
    GET.
    SET.

    DEFINE PUBLIC PROPERTY Address AS CHARACTER  NO-UNDO
    GET.
    SET.

    DEFINE PUBLIC PROPERTY Address2 AS CHARACTER  NO-UNDO
    GET.
    SET.

    DEFINE PUBLIC PROPERTY City AS CHARACTER  NO-UNDO
    GET.
    SET.

    DEFINE PUBLIC PROPERTY State AS CHARACTER  NO-UNDO
    GET.
    SET.
```

```
ROUTINE-LEVEL ON ERROR UNDO, THROW.

USING Demo.ManagingDataInObjects.SimpleValueObject.* FROM PROPATH .

DEFINE VARIABLE oCustomer AS Customer NO-UNDO .

FIND FIRST Customer .

oCustomer = NEW Customer () .

ASSIGN oCustomer:CustNum = Customer.CustNum
       oCustomer:Country = Customer.Country
       oCustomer:Address = Customer.Address
       oCustomer:Address2 = Customer.Address2
       oCustomer:City = Customer.City
       oCustomer:PostalCode = Customer.PostalCode
       oCustomer:State = Customer.State
       oCustomer:Balance = Customer.Balance
       oCustomer:Comments = Customer.Comments
       oCustomer:Contact = Customer.Contact
       oCustomer:CreditLimit = Customer.CreditLimit
       oCustomer:CustNum = Customer.CustNum
       oCustomer:Discount = Customer.Discount
       oCustomer:EmailAddress = Customer.EmailAddress
       oCustomer:Fax = Customer.Fax
       oCustomer:Name = Customer.Name
       oCustomer:Phone = Customer.Phone
       oCustomer:SalesRep = Customer.SalesRep
       oCustomer:Terms = Customer.Terms .
```

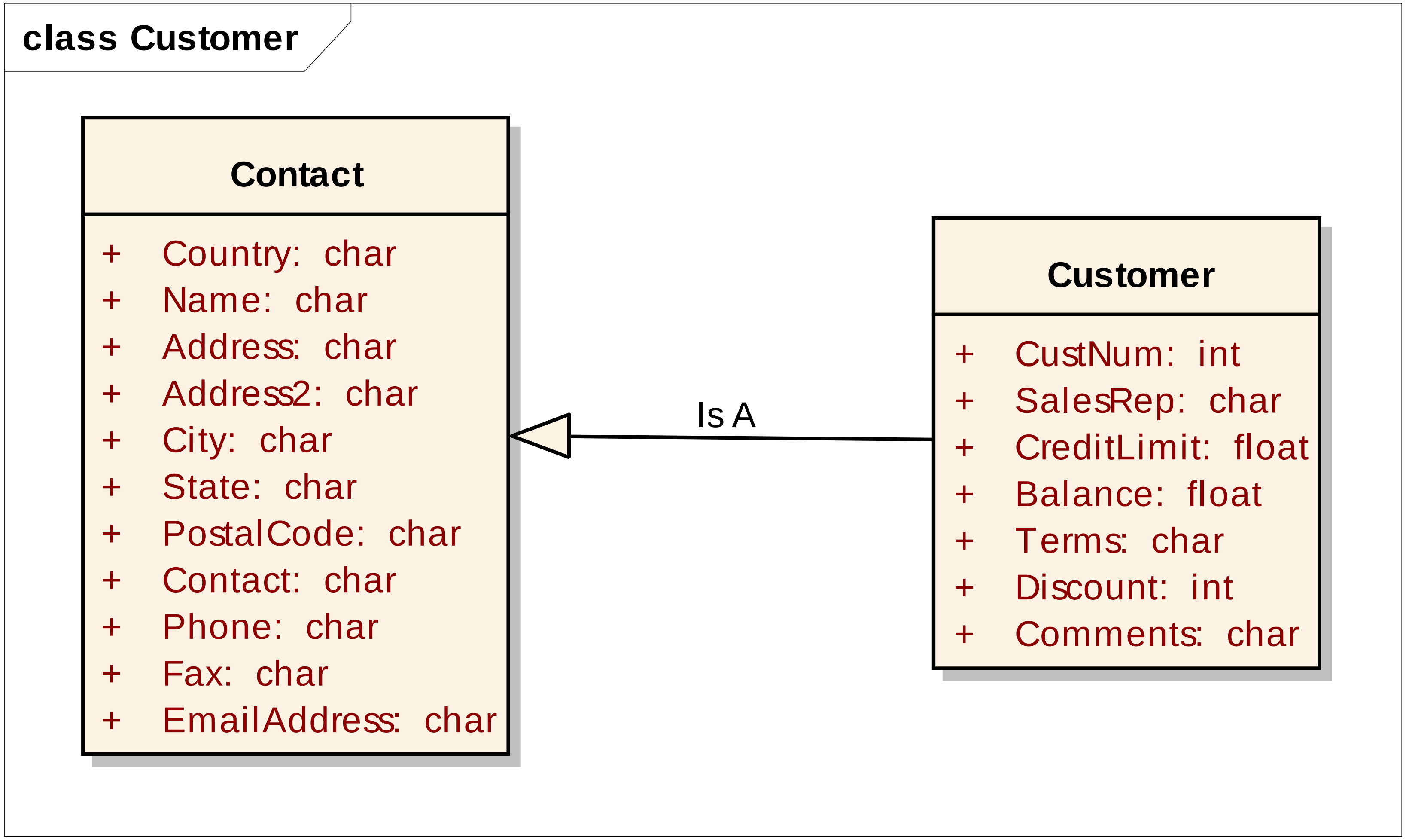
```
RUN DisplayCustomer (oCustomer) .

PROCEDURE DisplayCustomer:

    DEFINE INPUT PARAMETER oCustomer AS Customer NO-UNDO .

    MESSAGE oCustomer:CustNum SKIP
            oCustomer:Name    SKIP
            oCustomer:City
        VIEW-AS ALERT-BOX.

END.
```



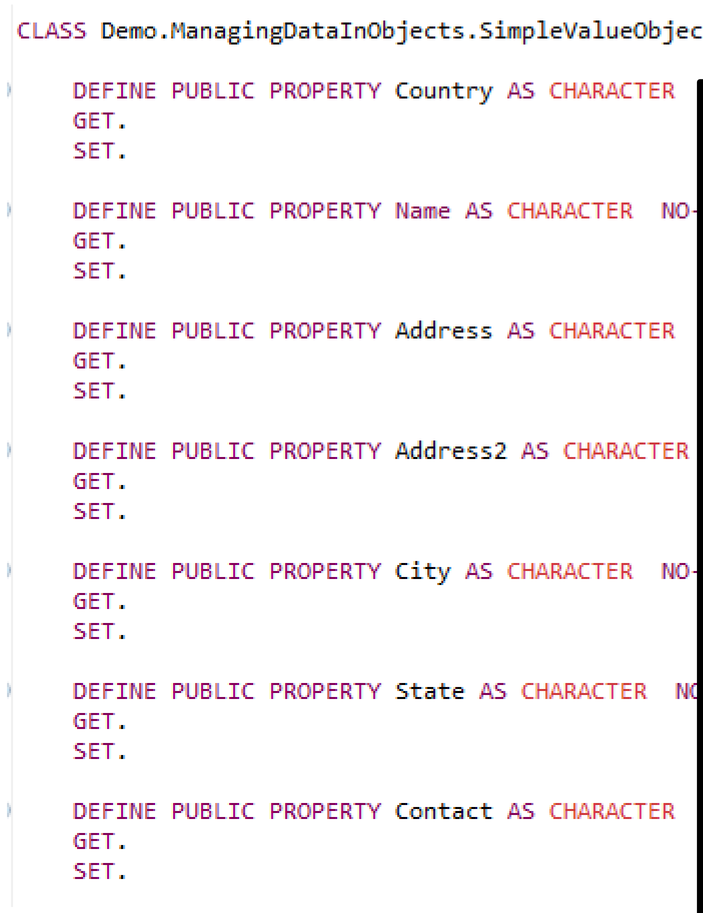
```
1
Lift Line Skiing Inc..
Burlington
```

- “Customer” has two meanings here. Not only you can get confused. The compiler as well…

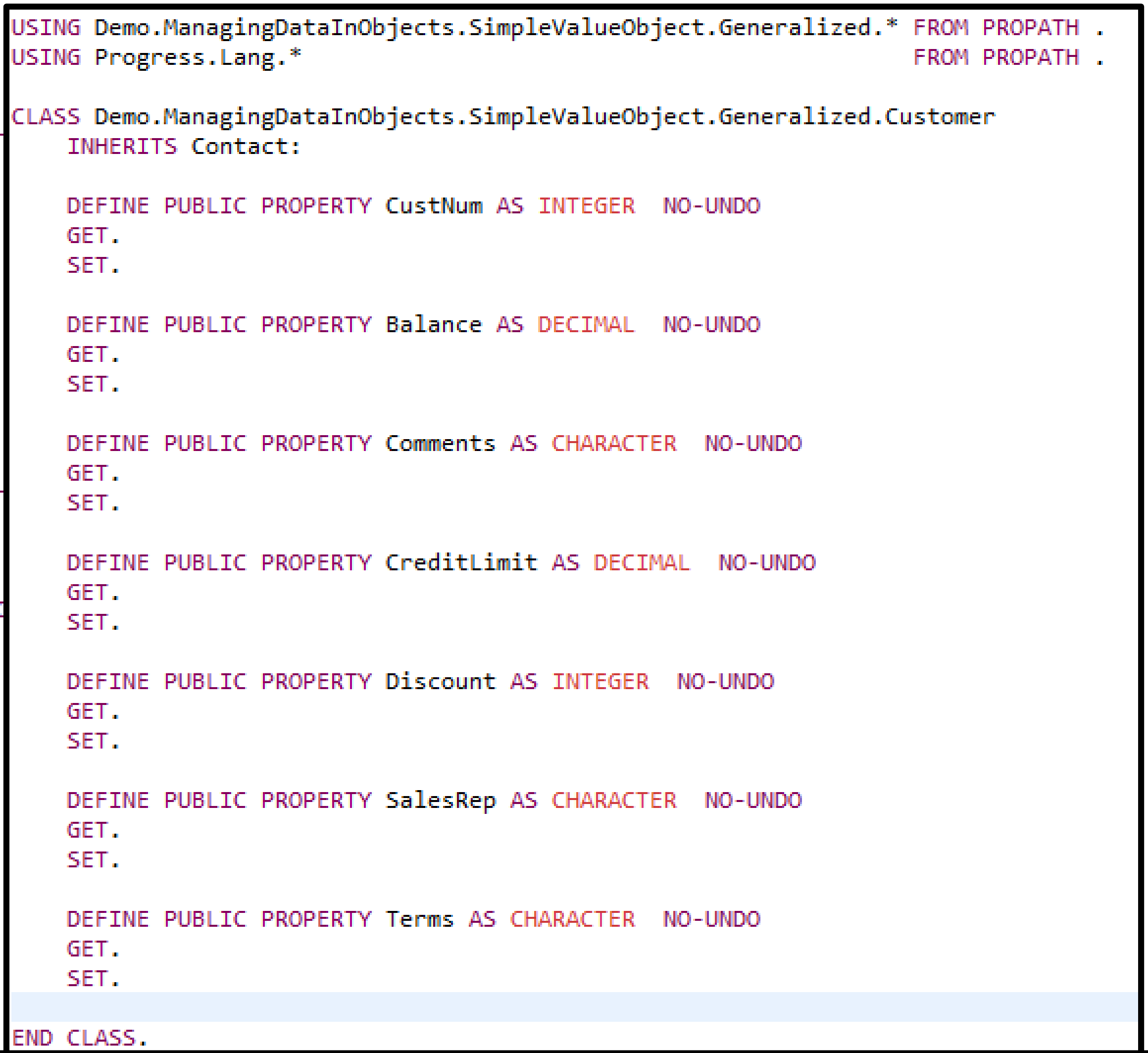
Generalization



Generalization



```
CLASS Demo.ManagingDataInObjects.SimpleValueObject.Generalized.Contact:

    DEFINE PUBLIC PROPERTY Country AS CHARACTER
    GET.
    SET.

    DEFINE PUBLIC PROPERTY Name AS CHARACTER  NO-
    GET.
    SET.

    DEFINE PUBLIC PROPERTY Address AS CHARACTER
    GET.
    SET.

    DEFINE PUBLIC PROPERTY Address2 AS CHARACTER
    GET.
    SET.

    DEFINE PUBLIC PROPERTY City AS CHARACTER  NO-
    GET.
    SET.

    DEFINE PUBLIC PROPERTY State AS CHARACTER  NO
    GET.
    SET.

    DEFINE PUBLIC PROPERTY Contact AS CHARACTER
    GET.
    SET.
```



```
USING Demo.ManagingDataInObjects.SimpleValueObject.Generalized.* FROM PROPATH .
USING Progress.Lang.*                                             FROM PROPATH .

CLASS Demo.ManagingDataInObjects.SimpleValueObject.Generalized.Customer
    INHERITS Contact:

    DEFINE PUBLIC PROPERTY CustNum AS INTEGER  NO-UNDO
    GET.
    SET.

    DEFINE PUBLIC PROPERTY Balance AS DECIMAL  NO-UNDO
    GET.
    SET.

    DEFINE PUBLIC PROPERTY Comments AS CHARACTER  NO-UNDO
    GET.
    SET.

    DEFINE PUBLIC PROPERTY CreditLimit AS DECIMAL  NO-UNDO
    GET.
    SET.

    DEFINE PUBLIC PROPERTY Discount AS INTEGER  NO-UNDO
    GET.
    SET.

    DEFINE PUBLIC PROPERTY SalesRep AS CHARACTER  NO-UNDO
    GET.
    SET.

    DEFINE PUBLIC PROPERTY Terms AS CHARACTER  NO-UNDO
    GET.
    SET.

END CLASS.
```

Generalization

- Consumer of “Customer” does not need to know that it is inheriting “Contact”
- “Contact” part may be reused for “Supplier”
- Code may expect “Contact” as a parameter and receive “Customer” or “Supplier” (send Email, write letter)

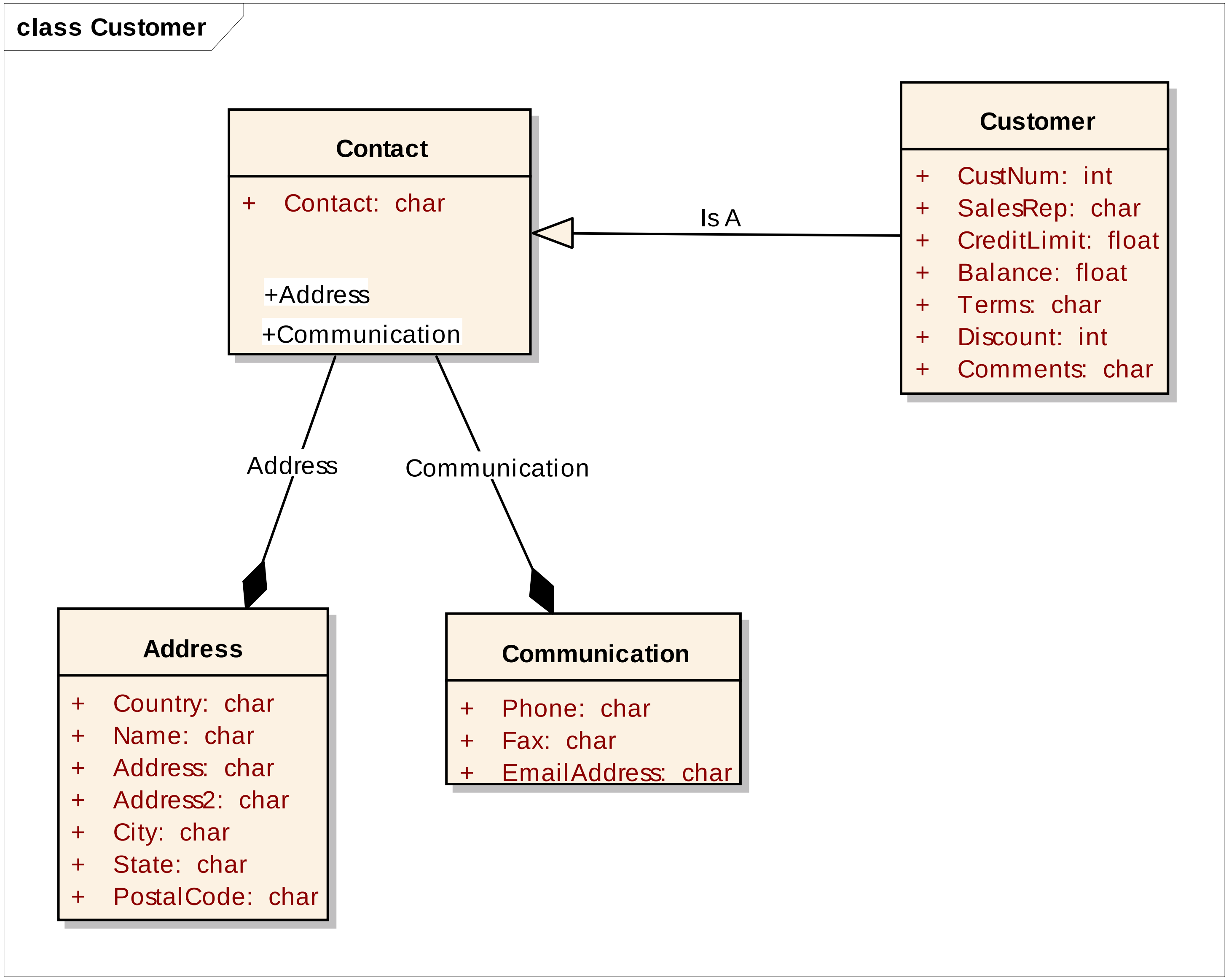
- Properties are always inherited (unless they are private)
- No property overloading, no ability to redefine

Agenda

- Rules Showcase
- Value Objects
- Object Relational Mapping
- Business Entities
- Dataset Model Classes
- Conclusion

Object Relational Mapping

- Value objects with Data Access …
- .NET: Entity Framework, NHibernate
- Java: Hibernate, Java Persistence API
- Abstracting Database structure in code to object optimized form
- Reversal of reference: In DB child records have the key of the parent, in objects parents typically have the reference to the child(s)


class Customer
Contact
+ Contact: char
+Address
+Communication
Customer
+ CustNum: int
+ SalesRep: char
+ CreditLimit: float
+ Balance: float
+ Terms: char
+ Discount: int
+ Comments: char
Is A
Address
Communication
Address
+ Country: char
+ Name: char
+ Address: char
+ Address2: char
+ City: char
+ State: char
+ PostalCode: char
Communication
+ Phone: char
+ Fax: char
+ EmailAddress: char

```
USING Demo.ManagingDataInObjects.Orm.Customer.* FROM PROPATH .

DEFINE VARIABLE oCustomer AS Customer NO-UNDO .

FIND FIRST Customer .

oCustomer = NEW Customer () .

ASSIGN oCustomer:CustNum = Customer.CustNum
       oCustomer:Balance = Customer.Balance
       oCustomer:Comments = Customer.Comments
       oCustomer:Contact = Customer.Contact
       oCustomer:CreditLimit = Customer.CreditLimit
       oCustomer:CustNum = Customer.CustNum
       oCustomer:Discount = Customer.Discount
       oCustomer:SalesRep = Customer.SalesRep
       oCustomer:Terms = Customer.Terms

       oCustomer:Address:Country = Customer.Country
       oCustomer:Address:Name = Customer.Name
       oCustomer:Address:Address = Customer.Address
       oCustomer:Address:Address2 = Customer.Address2
       oCustomer:Address:City = Customer.City
       oCustomer:Address:PostalCode = Customer.PostalCode
       oCustomer:Address:State = Customer.State

       oCustomer:Communication:EmailAddress = Customer.EmailAddress
       oCustomer:Communication:Fax = Customer.Fax
       oCustomer:Communication:Phone = Customer.Phone .
```

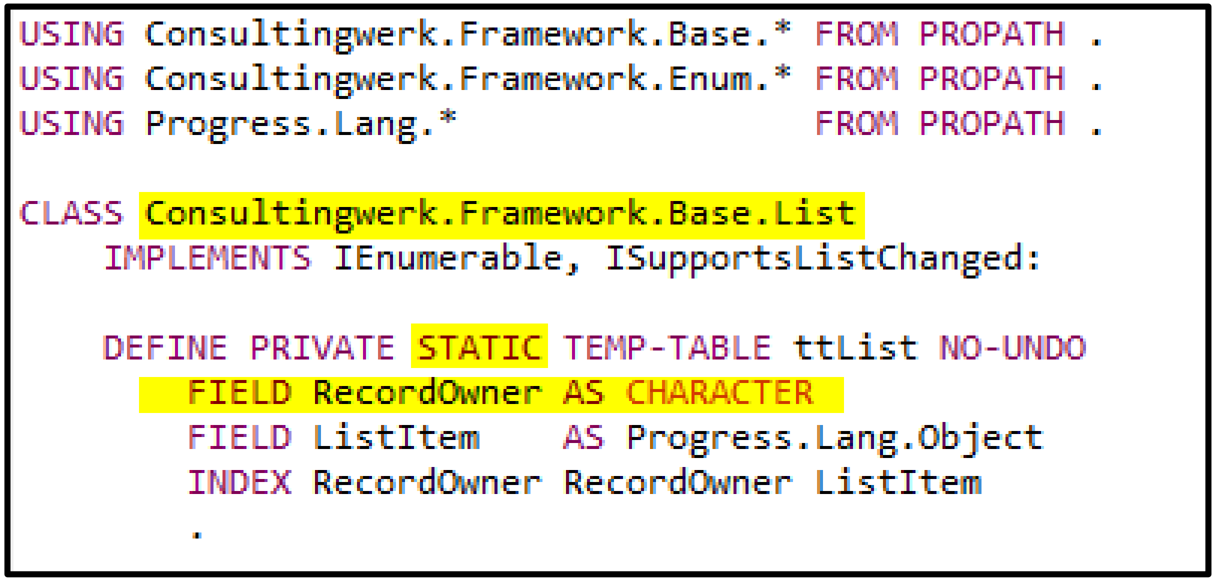
Multiplicity

- OOABL limited support for “lists” of objects
- Array of Objects
- Temp-Table with Progress.Lang.Object fields
  - Potential issues with DBI file size
- Linked Lists
  - Complex implementation

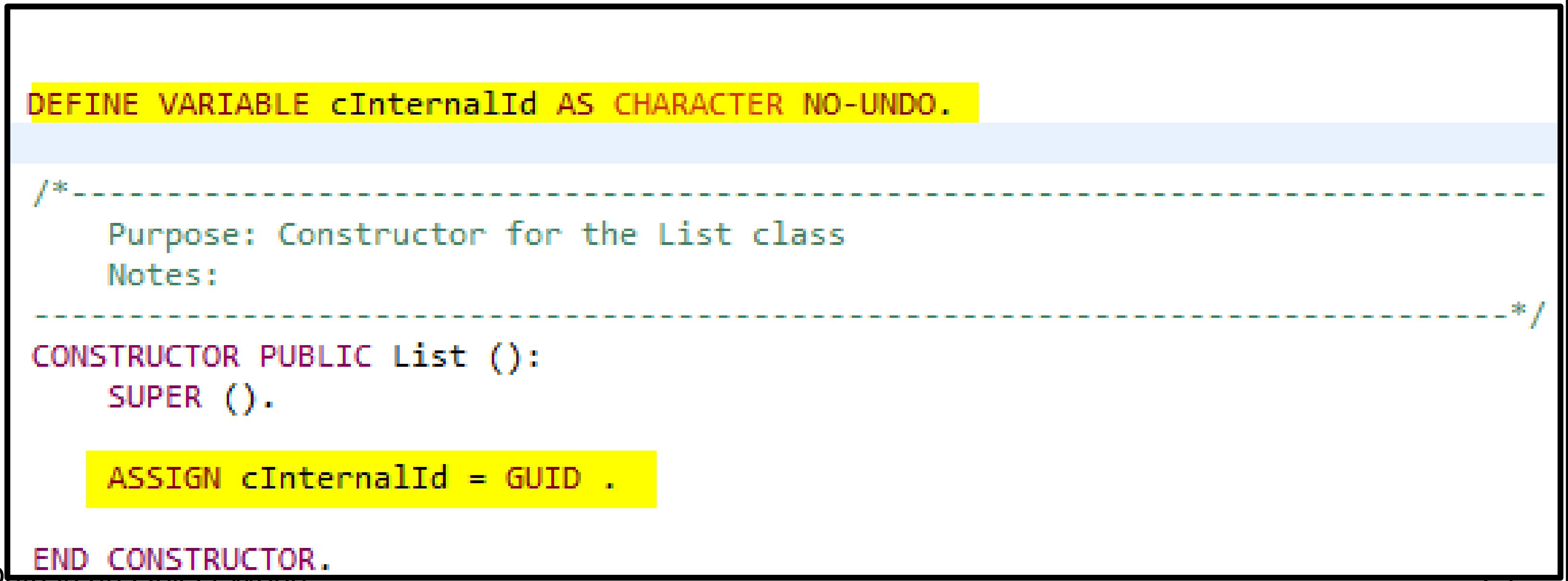
- Unless using Array’s you should wrap List implementation in separate classes for reuse and separation of concern

Reducing Temp-Table overhead

- Every temp-table (OE11 once it contains a single record) allocates 9 blocks in the DBI file
- Many small temp-tables blow up DBI fast
- Consider wrapping List code in class that uses a single temp-table for all instances of the List
- “static” temp-table (in OO means)



```
USING Consultingwerk.Framework.Base.* FROM PROPATH .
USING Consultingwerk.Framework.Enum.* FROM PROPATH .
USING Progress.Lang.*                 FROM PROPATH .

CLASS Consultingwerk.Framework.Base.List
    IMPLEMENTS IEnumerable, ISupportsListChanged:

    DEFINE PRIVATE STATIC TEMP-TABLE ttList NO-UNDO
        FIELD RecordOwner AS CHARACTER
        FIELD ListItem    AS Progress.Lang.Object
        INDEX RecordOwner RecordOwner ListItem
        .
```



```
DEFINE VARIABLE cInternalId AS CHARACTER NO-UNDO.

/*------------------------------------------------------------------------------
    Purpose: Constructor for the List class
    Notes:
------------------------------------------------------------------------------*/
CONSTRUCTOR PUBLIC List ():
    SUPER ().

    ASSIGN cInternalId = GUID .

END CONSTRUCTOR.
```

List:Add ()

```
/*------------------------------------------------------------------------------
    Purpose: Adds an Item to the List
    Notes:
    @param poItem The Item to add to the List
    @return The item that was added to the List
------------------------------------------------------------------------------*/
METHOD PUBLIC Progress.Lang.Object Add (poItem AS Progress.Lang.Object):

    DEFINE BUFFER ttList FOR ttList .

    CREATE ttList.
    ASSIGN ttList.RecordOwner = cInternalId
           ttList.ListItem    = poItem .

    THIS-OBJECT:OnListChanged (NEW ListChangedEventArgs (ListChangedTypeEnum:ListItemAdded)) .

    RETURN poItem .

END METHOD.
```

List:GetItem ()

```
/*------------------------------------------------------------------------------
    Purpose: Returns the Item at the specified Index
    Notes:
    @param piIndex The index of the Item to be returned
    @return The object at the specified list position
------------------------------------------------------------------------------*/
METHOD PUBLIC Progress.Lang.Object GetItem (piIndex AS INTEGER):

    DEFINE BUFFER ttList FOR ttList .

    DEFINE VARIABLE i AS INTEGER NO-UNDO.

    DO i = 1 TO piIndex:

        FIND NEXT ttList WHERE ttList.RecordOwner = cInternalId NO-ERROR .

        IF NOT AVAILABLE ttList THEN
            UNDO, THROW NEW AppError ("The specified index is not part of the List"{&TRAN}, 0) .

    END.

    IF AVAILABLE ttList THEN
        RETURN ttList.ListItem .

END METHOD.
```

Sample using List

```
DEFINE VARIABLE oList     AS Consultingwerk.Framework.Base.List NO-UNDO .
DEFINE VARIABLE oCustomer AS Customer NO-UNDO .

oList = NEW Consultingwerk.Framework.Base.List () .

FOR EACH Customer WHERE Customer.CustNum <= 5:

    oCustomer = CAST (oList:Add (NEW Customer()),
                      Customer) .

    ASSIGN oCustomer:CustNum = Customer.CustNum
           oCustomer:Balance = Customer.Balance
           oCustomer:Comments = Customer.Comments
           oCustomer:Contact = Customer.Contact
           oCustomer:CreditLimit = Customer.CreditLimit
           oCustomer:CustNum = Customer.CustNum
           oCustomer:Discount = Customer.Discount
           oCustomer:SalesRep = Customer.SalesRep
           oCustomer:Terms = Customer.Terms

           oCustomer:Address:Country = Customer.Country
           oCustomer:Address:Name = Customer.Name
           oCustomer:Address:Address = Customer.Address
```

Sample using List

```
PROCEDURE DisplayCustomer:

    DEFINE INPUT PARAMETER oList AS Consultingwerk.Framework.Base.List NO-UNDO .

    DEFINE VARIABLE oCustomer AS Customer NO-UNDO .
    DEFINE VARIABLE i         AS INTEGER  NO-UNDO.

    DO i = 1 TO oList:Count:

        oCustomer = CAST (oList:GetItem (i), Customer) .

        MESSAGE oCustomer:CustNum          SKIP
                oCustomer:Address:Name     SKIP
                oCustomer:Address:City
            VIEW-AS ALERT-BOX.

    END.
END.
```

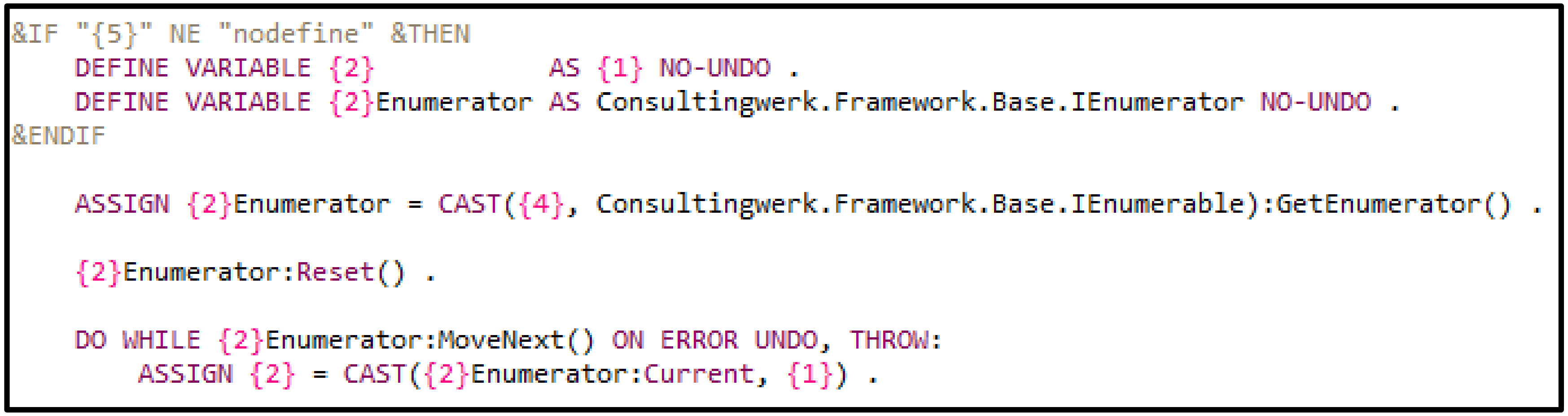
Sample using Enumerator

- New “Statement” using Include File

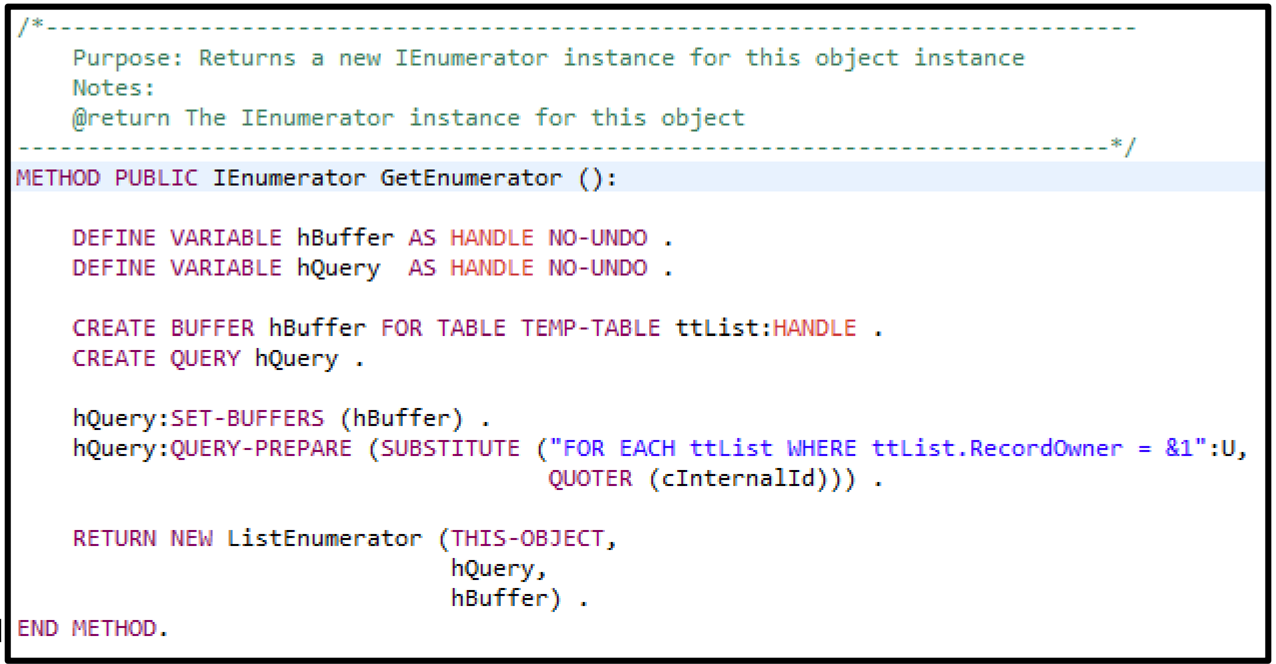
```
PROCEDURE DisplayCustomer:

    DEFINE INPUT PARAMETER oList AS Consultingwerk.Framework.Base.List NO-UNDO .

    {Consultingwerk/foreachABL.i Customer oCustomer in oList}

        MESSAGE oCustomer:CustNum          SKIP
                oCustomer:Address:Name     SKIP
                oCustomer:Address:City
            VIEW-AS ALERT-BOX.

    END.
END.
```

- List needs to support IEnumerable Interface

foreachABL.i



```
&IF "{5}" NE "nodefine" &THEN
    DEFINE VARIABLE {2}           AS {1} NO-UNDO .
    DEFINE VARIABLE {2}Enumerator AS Consultingwerk.Framework.Base.IEnumerator NO-UNDO .
&ENDIF

    ASSIGN {2}Enumerator = CAST({4}, Consultingwerk.Framework.Base.IEnumerable):GetEnumerator() .

    {2}Enumerator:Reset() .

    DO WHILE {2}Enumerator:MoveNext() ON ERROR UNDO, THROW:
        ASSIGN {2} = CAST({2}Enumerator:Current, {1}) .
```

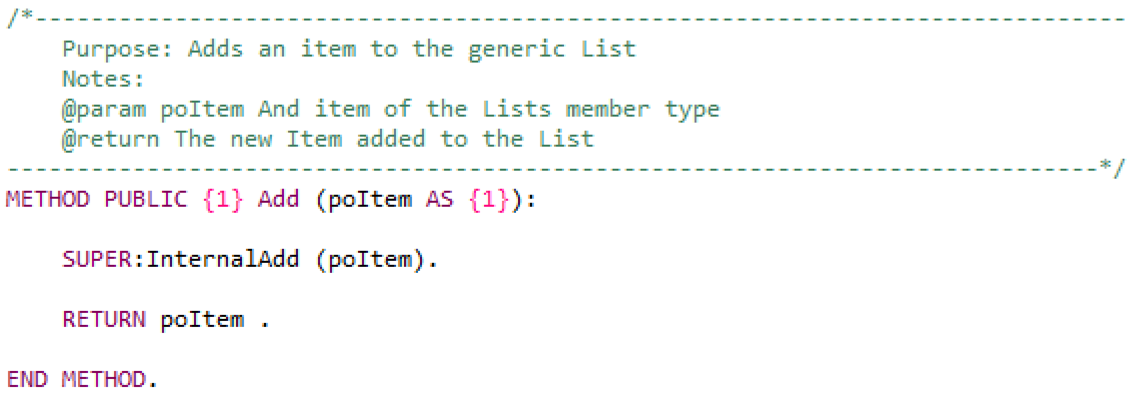


```
/*------------------------------------------------------------------------------
    Purpose: Returns a new IEnumerator instance for this object instance
    Notes:
    @return The IEnumerator instance for this object
------------------------------------------------------------------------------*/
METHOD PUBLIC IEnumerator GetEnumerator ():

    DEFINE VARIABLE hBuffer AS HANDLE NO-UNDO .
    DEFINE VARIABLE hQuery  AS HANDLE NO-UNDO .

    CREATE BUFFER hBuffer FOR TABLE TEMP-TABLE ttList:HANDLE .
    CREATE QUERY hQuery .

    hQuery:SET-BUFFERS (hBuffer) .
    hQuery:QUERY-PREPARE (SUBSTITUTE ("FOR EACH ttList WHERE ttList.RecordOwner = &1":U,
                                      QUOTER (cInternalId))) .

    RETURN NEW ListEnumerator (THIS-OBJECT,
                               hQuery,
                               hBuffer) .

END METHOD.
```

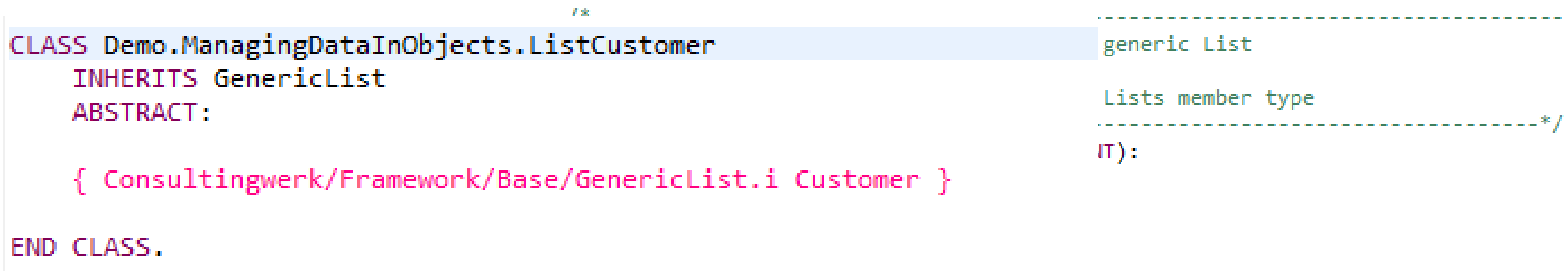
Generic Lists

- Base List class Add method allows Progress.Lang.Object, i.o.W. every kind of member
- Return value of GetItem needs CAST
- A specialized List class of “Customer” guaranties that only customer objects are part of the List
- .NET has dynamic language concepts for this: Generic lists: List<Customer>
- ABL needs class for every list. **Don’t be shy and use Include Files in Classes!**

Generic List

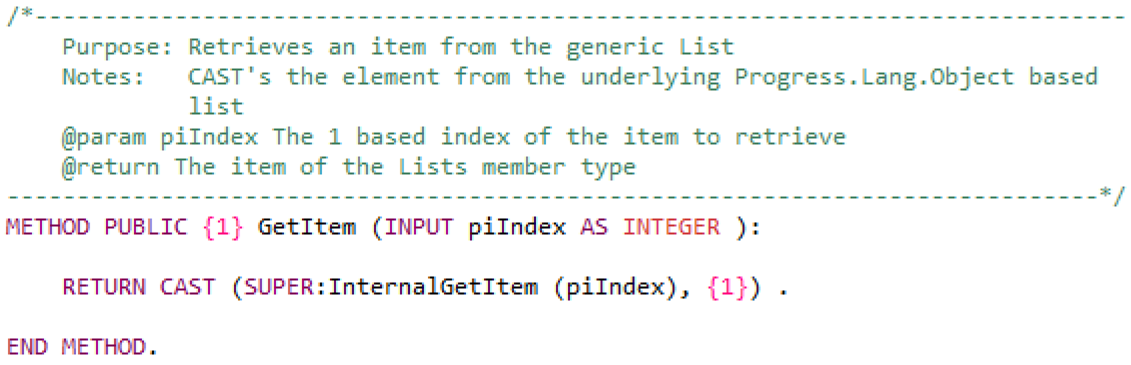


```
/*------------------------------------------------------------------------------
    Purpose: Adds an item to the generic List
    Notes:
    @param poItem And item of the Lists member type
    @return The new Item added to the List
------------------------------------------------------------------------------*/
METHOD PUBLIC {1} Add (poItem AS {1}):

    SUPER:InternalAdd (poItem).

    RETURN poItem .

END METHOD.
```



```
CLASS Demo.ManagingDataInObjects.ListCustomer
    INHERITS GenericList
    ABSTRACT:

    { Consultingwerk/Framework/Base/GenericList.i Customer }

END CLASS.
```

```
/*
--------------------------------------
generic List

Lists member type
-------------------------------------*/
IT):
```



```
/*------------------------------------------------------------------------------
    Purpose: Retrieves an item from the generic List
    Notes:   CAST's the element from the underlying Progress.Lang.Object based
             list
    @param piIndex The 1 based index of the item to retrieve
    @return The item of the Lists member type
------------------------------------------------------------------------------*/
METHOD PUBLIC {1} GetItem (INPUT piIndex AS INTEGER ):

    RETURN CAST (SUPER:InternalGetItem (piIndex), {1}) .

END METHOD.
```

Filtering

- .NET has LINQ Support
- Language Integrated Natural Query
- Iterating a list of objects with filtering
- True FOR EACH on objects
- Difficult to achieve with objects in Progress
- You end up iterating the whole list and verifying condition on every object in loop
- Ungly code, difficult to debug

Consultingwerk
software architecture and development

What's missing

- Data Access: Our preferred way are Business Entities, when ORM view is abstracted from relational view, mapping code is required
- Before Image Handling: Transaction undo, error handling, optimistic locking
- Filtering
- UI Binding: ABL supports this for Temp-Tables
- AppServer boundary… require custom serialization
- A code generator ☺

Agenda

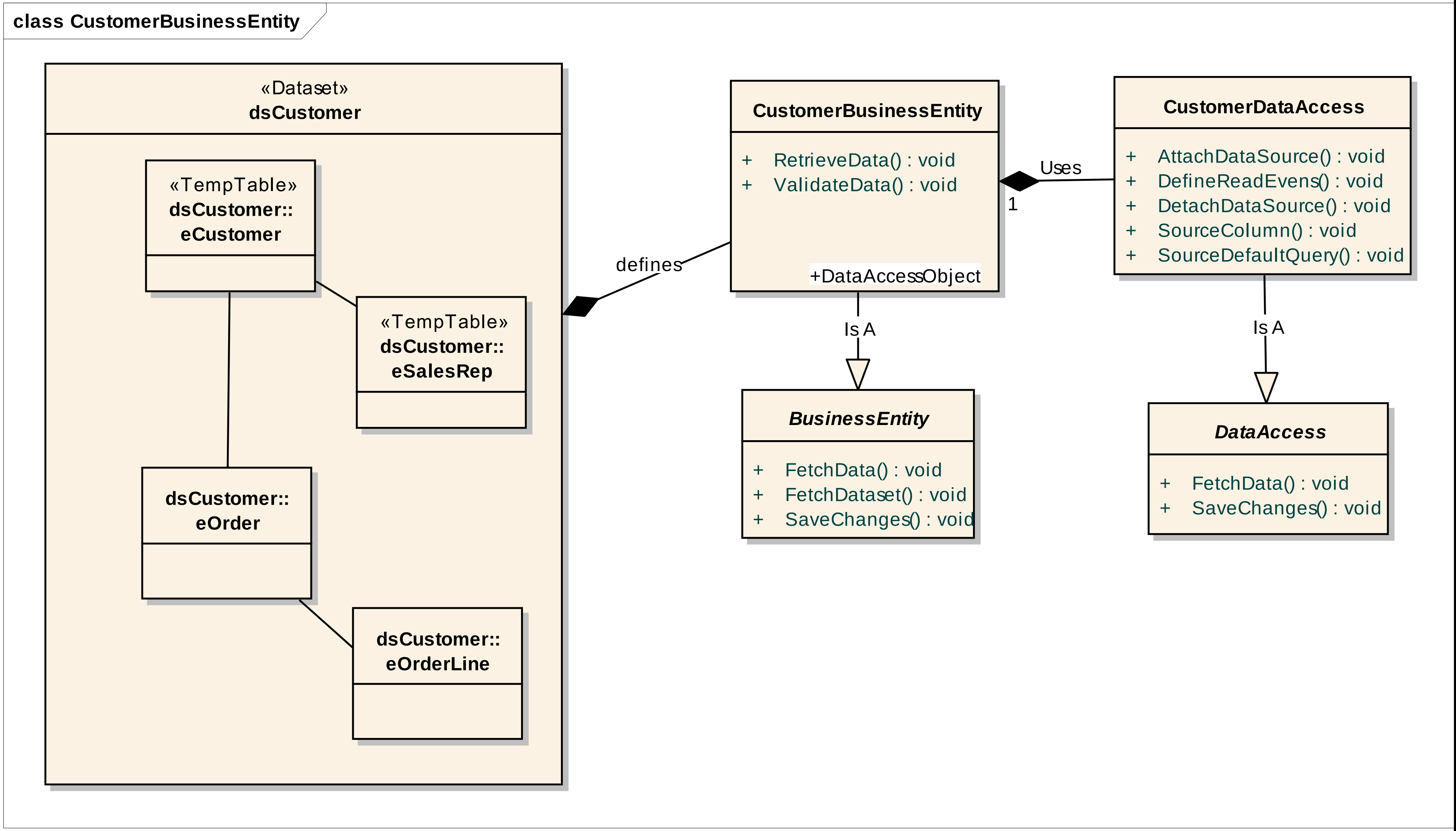
- Rules Showcase
- Value Objects
- Object Relational Mapping
- Business Entities
- Dataset Model Classes
- Conclusion

Business Entity

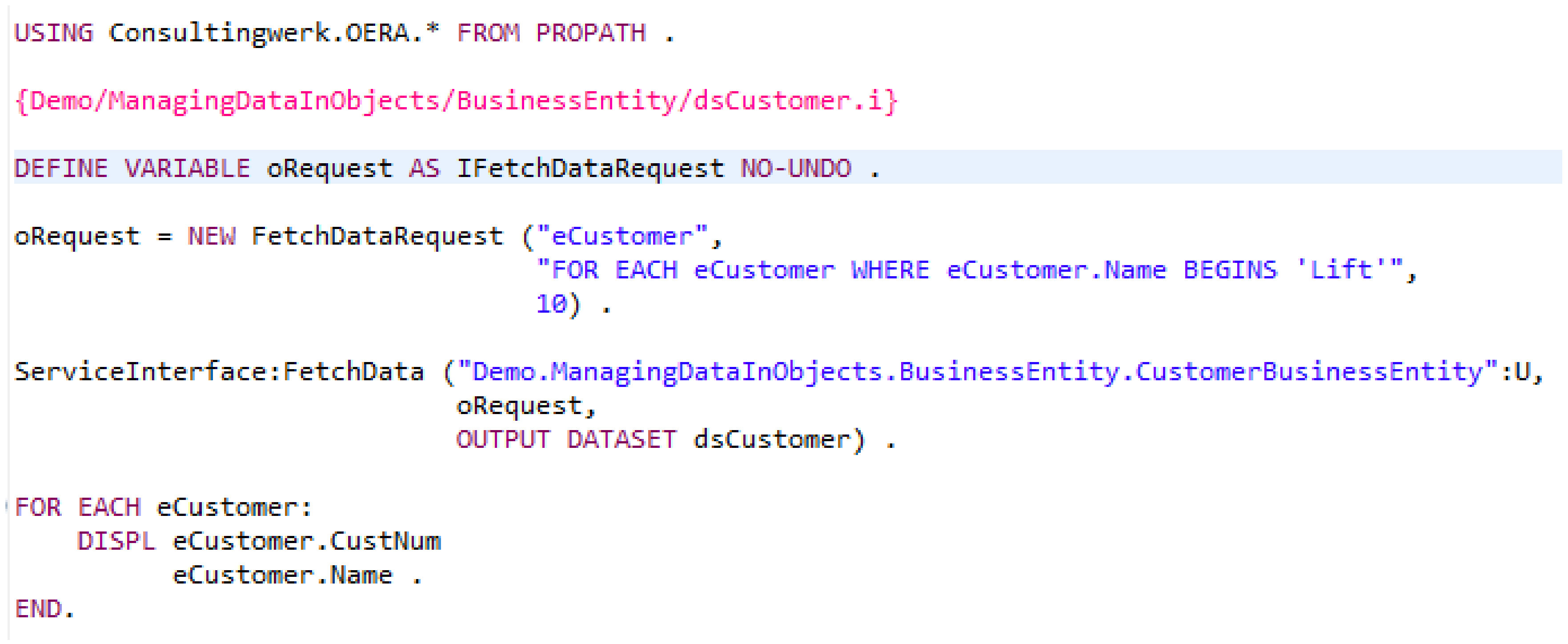
- Central term of the OERA
- OpenEdge Reference Architecture
- Business Logic Object typically build around a ProDataset
- Uses Data Access Object for reading and updating Data in the Database (or another storage)
- Term Business Entity abused in OE Mobile

Business Entity

- Business Entity: State Less Service Object (on the AppServer)
- ProDataset may be exposed to the outside
- ProDataset may be send to AppServer client
- Leveraging ProDataset before tables for optimistic locking and rollback
- Abstraction of the Dataset model possible in ProDataset design: Temp-Tables do not need to copy DB tables (completely), denormalization possible

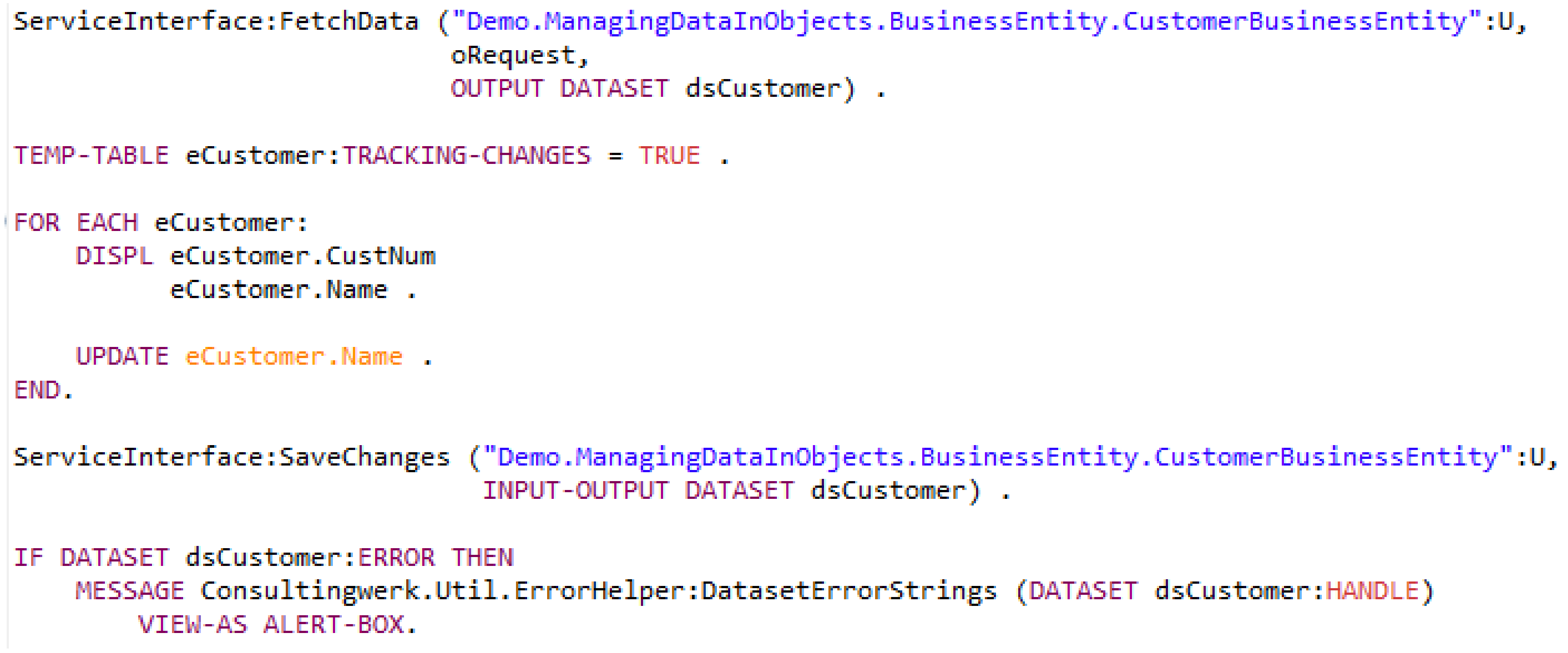

class CustomerBusinessEntity
«Dataset»
dsCustomer
«TempTable»
dsCustomer::
eCustomer
«TempTable»
dsCustomer::
eSalesRep
dsCustomer::
eOrder
dsCustomer::
eOrderLine
defines
CustomerBusinessEntity
+ RetrieveData() : void
+ ValidateData() : void
+DataAccessObject
Uses
1
CustomerDataAccess
+ AttachDataSource() : void
+ DefineReadEvens() : void
+ DetachDataSource() : void
+ SourceColumn() : void
+ SourceDefaultQuery() : void
Is A
Is A
BusinessEntity
+ FetchData() : void
+ FetchDataset() : void
+ SaveChanges() : void
DataAccess
+ FetchData() : void
+ SaveChanges() : void

Retrieving Data from a Business Entity



```
USING Consultingwerk.OERA.* FROM PROPATH .

{Demo/ManagingDataInObjects/BusinessEntity/dsCustomer.i}

DEFINE VARIABLE oRequest AS IFetchDataRequest NO-UNDO .

oRequest = NEW FetchDataRequest ("eCustomer",
                                 "FOR EACH eCustomer WHERE eCustomer.Name BEGINS 'Lift'",
                                 10) .

ServiceInterface:FetchData ("Demo.ManagingDataInObjects.BusinessEntity.CustomerBusinessEntity":U,
                            oRequest,
                            OUTPUT DATASET dsCustomer) .

FOR EACH eCustomer:
    DISPL eCustomer.CustNum
          eCustomer.Name .
END.
```

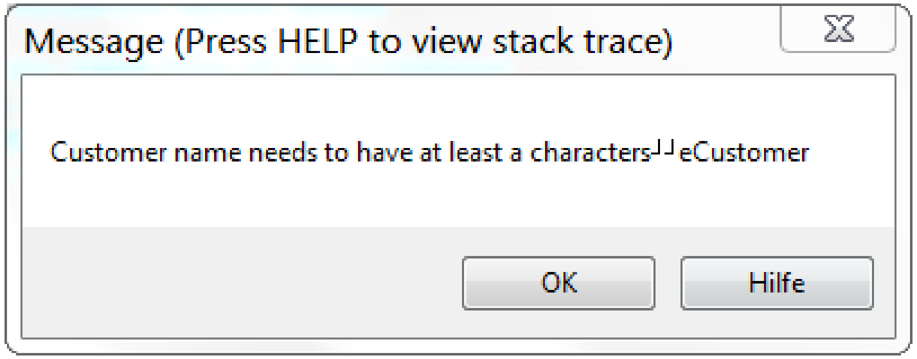
Updating Data using a Business Entity



```
ServiceInterface:FetchData ("Demo.ManagingDataInObjects.BusinessEntity.CustomerBusinessEntity":U,
                            oRequest,
                            OUTPUT DATASET dsCustomer) .

TEMP-TABLE eCustomer:TRACKING-CHANGES = TRUE .

FOR EACH eCustomer:
    DISPL eCustomer.CustNum
          eCustomer.Name .

    UPDATE eCustomer.Name .
END.

ServiceInterface:SaveChanges ("Demo.ManagingDataInObjects.BusinessEntity.CustomerBusinessEntity":U,
                              INPUT-OUTPUT DATASET dsCustomer) .

IF DATASET dsCustomer:ERROR THEN
    MESSAGE Consultingwerk.Util.ErrorHelper:DatasetErrorStrings (DATASET dsCustomer:HANDLE)
        VIEW-AS ALERT-BOX.
```

Business Entity Validation

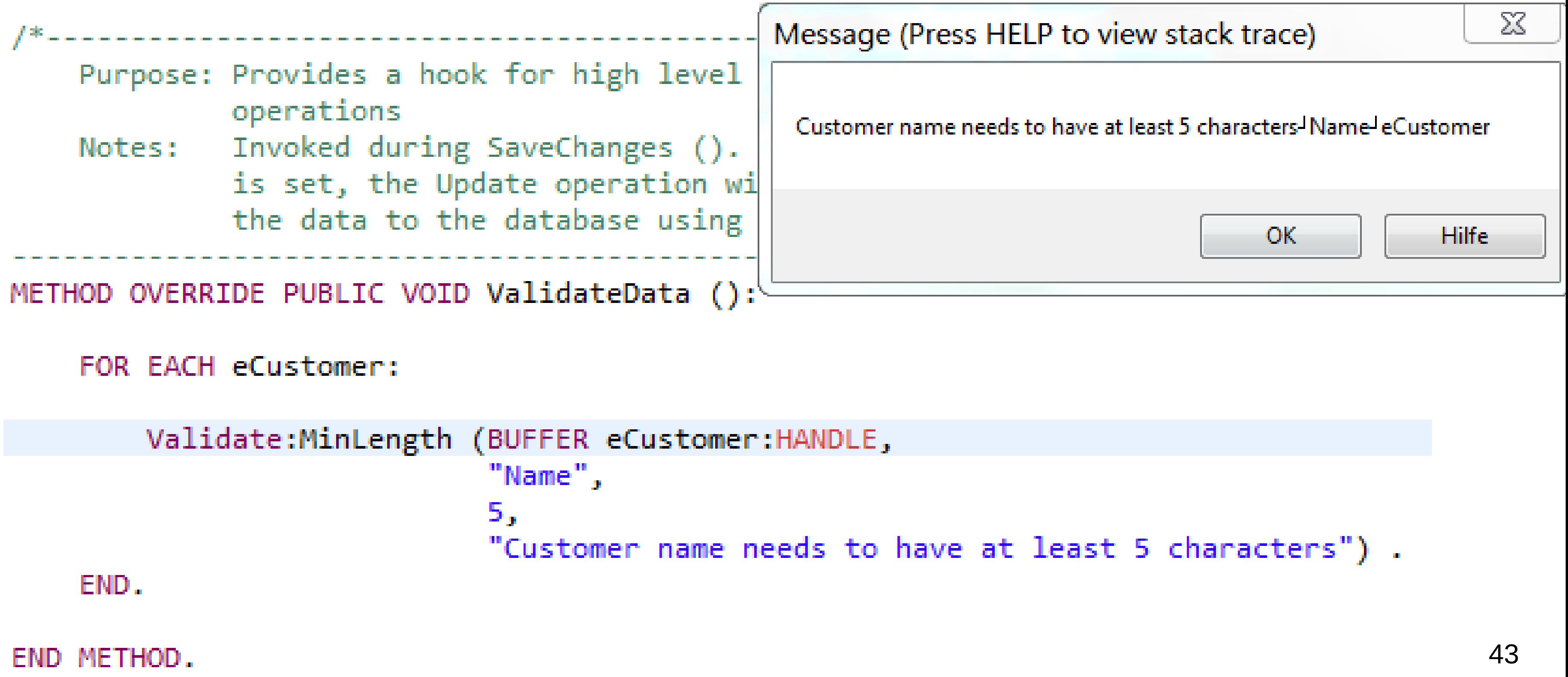
```
/*------------------------------------------------------------------------------
    Purpose: Provides a hook for high level data validation before Update
             operations
    Notes:   Invoked during SaveChanges (). When the ERROR flag of the ProDataset
             is set, the Update operation will be cancelled before writing back
             the data to the database using the DataAccess object
------------------------------------------------------------------------------*/
METHOD OVERRIDE PUBLIC VOID ValidateData ():

    FOR EACH eCustomer:

        IF LENGTH (eCustomer.Name) < 5 THEN
            ASSIGN BUFFER eCustomer:ERROR-STRING = "Customer name needs to have at least a characters"
                   BUFFER eCustomer:ERROR        = TRUE
                   DATASET dsCustomer:ERROR      = TRUE .
    END.

END METHOD.
```


Message (Press HELP to view stack trace)

Customer name needs to have at least a characters┘┘eCustomer

Validate API

- Assertion style validation
- Easier to read, don’t focus on error handling, include field info with error



```
/*------------------------------------------------------
    Purpose: Provides a hook for high level
             operations
    Notes:   Invoked during SaveChanges ().
             is set, the Update operation wi
             the data to the database using
--------------------------------------------------------
METHOD OVERRIDE PUBLIC VOID ValidateData ():

    FOR EACH eCustomer:

        Validate:MinLength (BUFFER eCustomer:HANDLE,
                            "Name",
                            5,
                            "Customer name needs to have at least 5 characters") .
    END.

END METHOD.
```

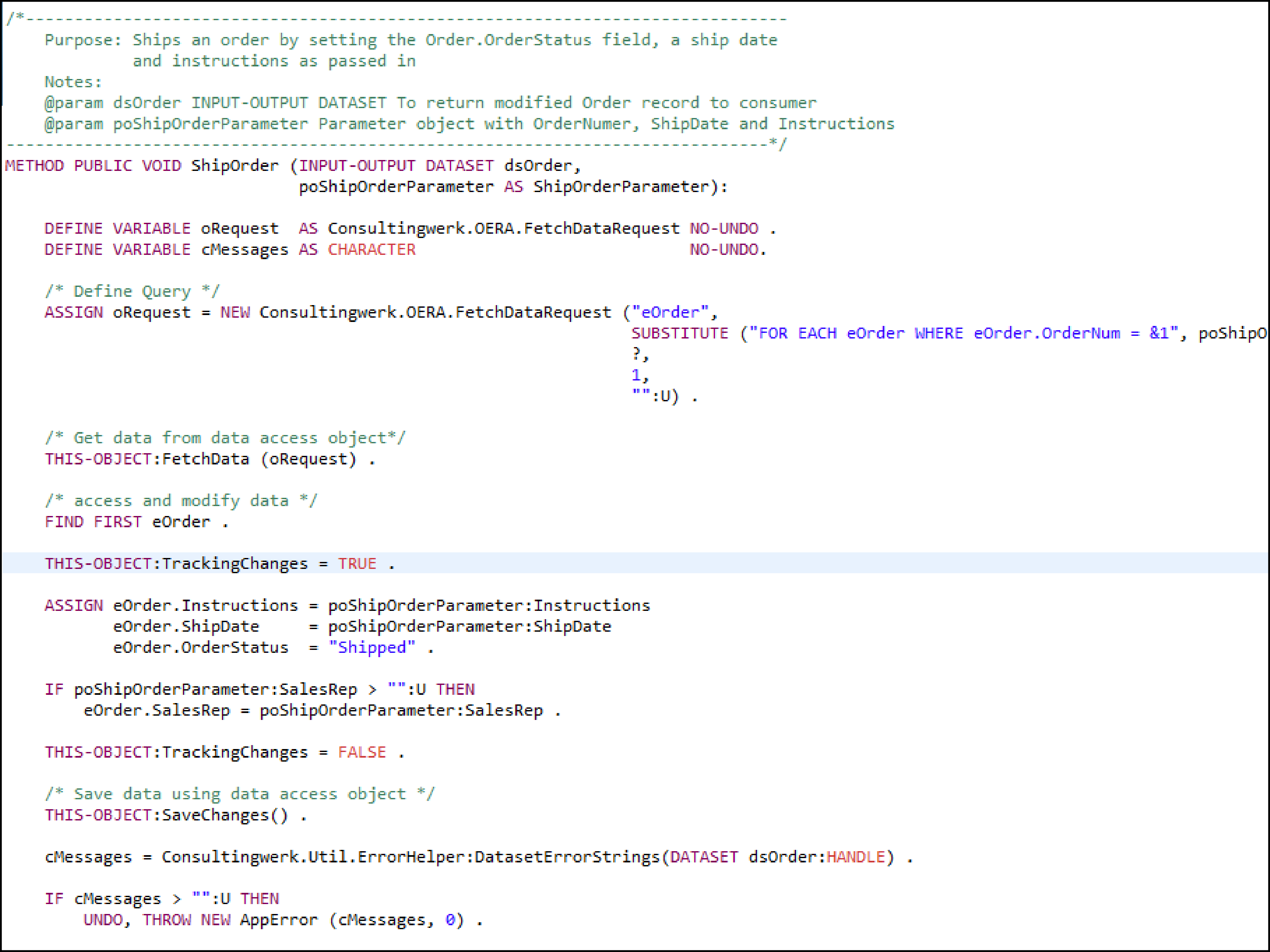
Message (Press HELP to view stack trace)

Customer name needs to have at least 5 characters┘Name┘eCustomer

OK Hilfe

Business Entity vs. Data Access Object

- Business Entity should contain Business Rules
- Data Access should contain Data Access ☺
- BE: Validation based on Business Rules
- DA: Validation based on DB schema, e.g. violating unique constraints, required foreing keys assigned, etc.
- BE: Should contain calculated values assignment
- BE: Should contain additional methods on the data, e.g. “ShipOrder”, “CancelDelivery”



```
/*------------------------------------------------------------------------------
    Purpose: Ships an order by setting the Order.OrderStatus field, a ship date
             and instructions as passed in
    Notes:
    @param dsOrder INPUT-OUTPUT DATASET To return modified Order record to consumer
    @param poShipOrderParameter Parameter object with OrderNumer, ShipDate and Instructions
------------------------------------------------------------------------------*/
METHOD PUBLIC VOID ShipOrder (INPUT-OUTPUT DATASET dsOrder,
                              poShipOrderParameter AS ShipOrderParameter):

    DEFINE VARIABLE oRequest  AS Consultingwerk.OERA.FetchDataRequest NO-UNDO .
    DEFINE VARIABLE cMessages AS CHARACTER                            NO-UNDO.

    /* Define Query */
    ASSIGN oRequest = NEW Consultingwerk.OERA.FetchDataRequest ("eOrder",
                                                                SUBSTITUTE ("FOR EACH eOrder WHERE eOrder.OrderNum = &1", poShipO
                                                                ?,
                                                                1,
                                                                "":U) .

    /* Get data from data access object*/
    THIS-OBJECT:FetchData (oRequest) .

    /* access and modify data */
    FIND FIRST eOrder .

    THIS-OBJECT:TrackingChanges = TRUE .

    ASSIGN eOrder.Instructions = poShipOrderParameter:Instructions
           eOrder.ShipDate     = poShipOrderParameter:ShipDate
           eOrder.OrderStatus  = "Shipped" .

    IF poShipOrderParameter:SalesRep > "":U THEN
        eOrder.SalesRep = poShipOrderParameter:SalesRep .

    THIS-OBJECT:TrackingChanges = FALSE .

    /* Save data using data access object */
    THIS-OBJECT:SaveChanges() .

    cMessages = Consultingwerk.Util.ErrorHelper:DatasetErrorStrings(DATASET dsOrder:HANDLE) .

    IF cMessages > "":U THEN
        UNDO, THROW NEW AppError (cMessages, 0) .
```

Demo

- Build Business Entity in Business Entity Designer
- Business Entity Tester
- Review code

Business Entity conclusion

- Build around first class citizen in the ABL: ProDataset
- Exposes ProDataset to the consumer
- Typically does not persist state information
- So, ProDataset is considered part of message to Business Entity, not violating encapsulation
- May be Data Provider for ORM
- ProDatasets supported with EVERY AppServer client: ABL, .NET, Java, Web Services, REST, OE Mobile, Rollbase

Agenda

- Rules Showcase
- Value Objects
- Object Relational Mapping
- Business Entities
- Dataset Model Classes
- Conclusion

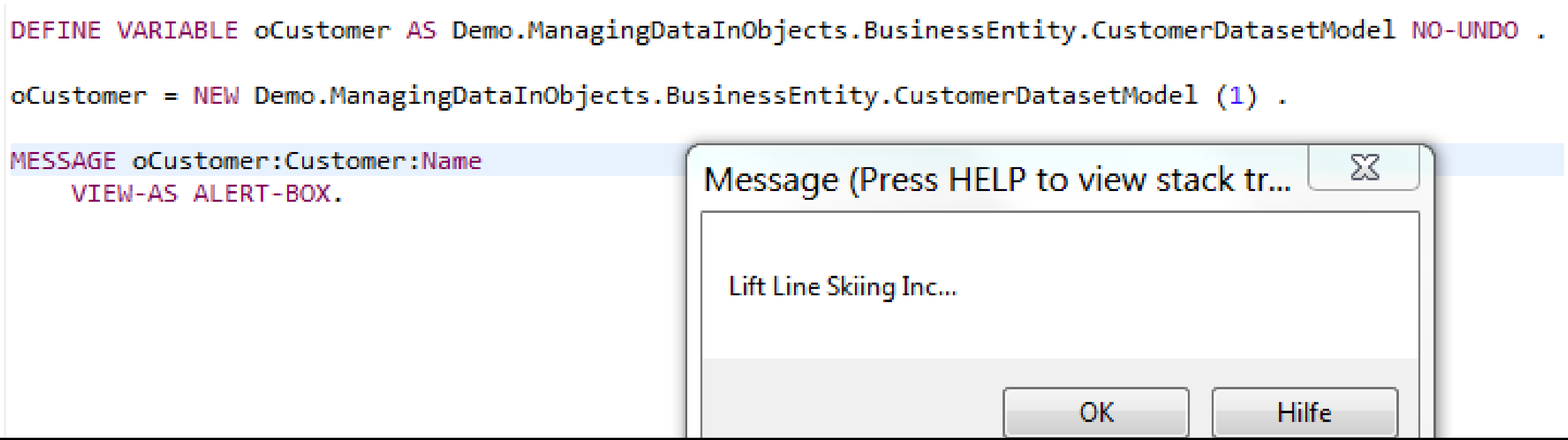
Dataset Model Classes

- Primary goal: Simplify access to Business Entities on server side code and clients, make it transparent if on client or AppServer
- Secondary goal: Provide more control on access to fields and tables
- Third goal: Neat query interface

- Build around ProDataset
- Typically build as consumer to a Business Entity
- Support for Read and Write

Dataset Model Classes

- Access to fields (and tables) through
- Main difference to ORM approach is, that there is only a single instance of the model that allows access to multiple records though iterators
- Model Class constructors may retrieve data using default (PUK) queries



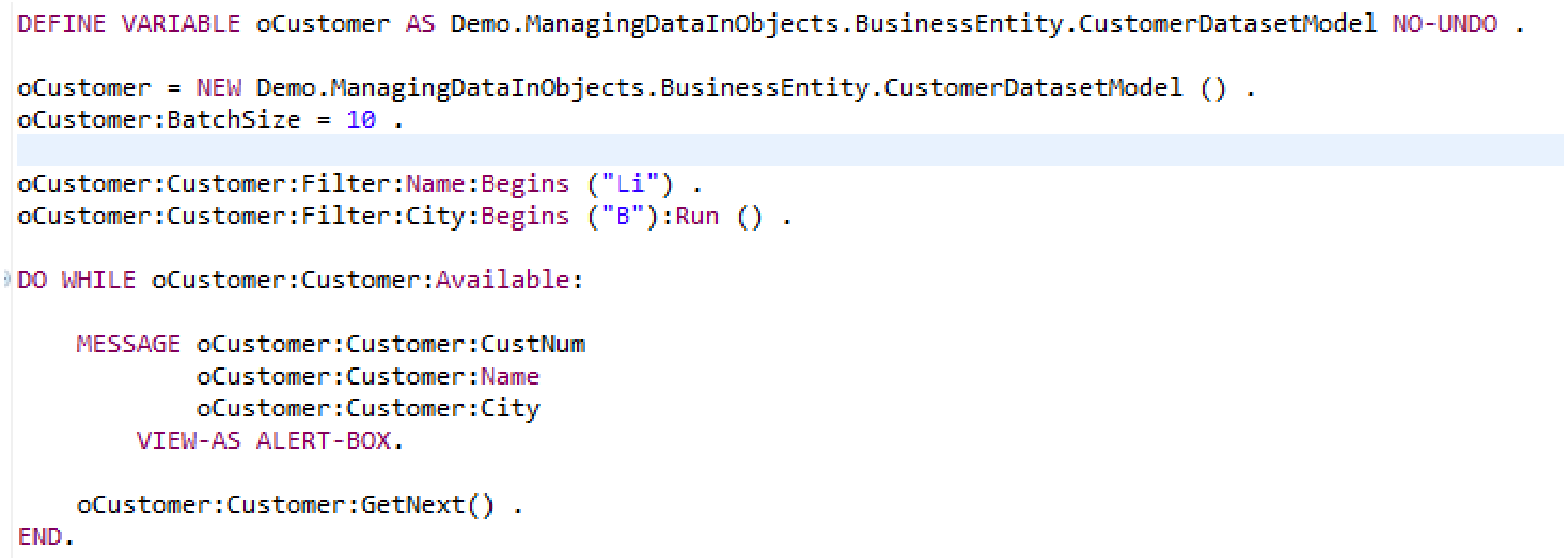
```
DEFINE VARIABLE oCustomer AS Demo.ManagingDataInObjects.BusinessEntity.CustomerDatasetModel NO-UNDO .

oCustomer = NEW Demo.ManagingDataInObjects.BusinessEntity.CustomerDatasetModel (1) .

MESSAGE oCustomer:Customer:Name
    VIEW-AS ALERT-BOX.
```

Message (Press HELP to view stack tr...

Lift Line Skiing Inc...

Query Sample

- Every “Table” class has a filter class attached
- Provides simple and strong typed query capabilities
- Batching support



```
DEFINE VARIABLE oCustomer AS Demo.ManagingDataInObjects.BusinessEntity.CustomerDatasetModel NO-UNDO .

oCustomer = NEW Demo.ManagingDataInObjects.BusinessEntity.CustomerDatasetModel () .
oCustomer:BatchSize = 10 .


oCustomer:Customer:Filter:Name:Begins ("Li") .
oCustomer:Customer:Filter:City:Begins ("B"):Run () .

DO WHILE oCustomer:Customer:Available:

    MESSAGE oCustomer:Customer:CustNum
            oCustomer:Customer:Name
            oCustomer:Customer:City
        VIEW-AS ALERT-BOX.

    oCustomer:Customer:GetNext() .
END.
```

Query Support

- We are using the same Filter techniques on top of data in the models
- Support for retrieving large batch of data from the Database and AppServer and run local sub queries
- Support for Viewes

Demo

- Review Dataset Model Class
- Execute Query
- Modify Data
- Build and Invoke Custom Method

Agenda

- Rules Showcase
- Value Objects
- Object Relational Mapping
- Business Entities
- Dataset Model Classes
- Conclusion

Conclusion

- OOABL is ready for Data Access!
- Different approaches
  - there is no single right answer
  - some may be more ABL style than others which may be more Java or C# style
- Don’t ban ABL language element from your code (include files, ProDataset, TempTable)
- Pick YOURS!
- **Any approach will require building foundation classes, consider code generation**

Questions?