An OO Code Generator
A Live OO Project

Part I

PUG Challenge Americas 2017

Tim Kuehn

Senior Consultant

www.tdkcs.com

Outline

Presentation Goal:
To discuss the application of various OO concepts in a real-world project that does something useful.

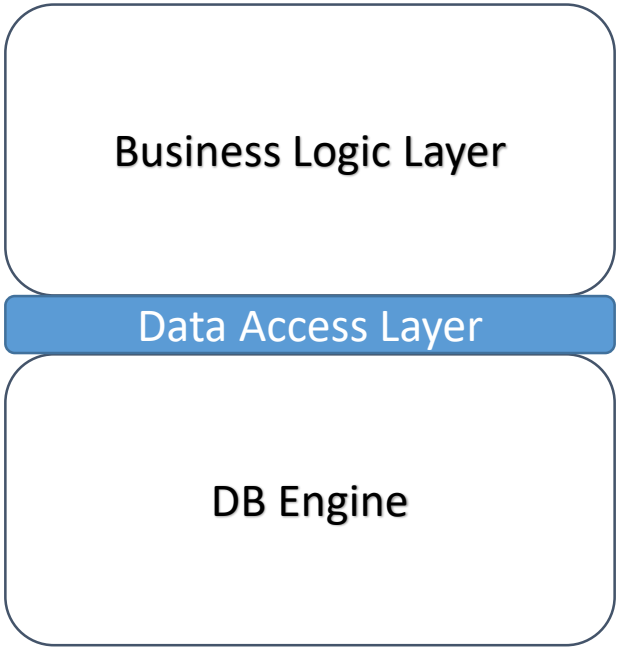
Project Goal:
Design and implement a system that'll automate the creation of various data-access classes, TT definitions, PDS definitions, and data sources.

Development Process:
- Write code,
- Get it working,
- Refactor to manage scope of concern and eliminate duplicated functionality,
- Lather, Rinse, Repeat.

Note:
This code is a ‘work in progress’ and is for instructional purposes only.
Project code can be downloaded from https://bitbucket.org/TDKConsulting/

Project Goal

In an ideal world….

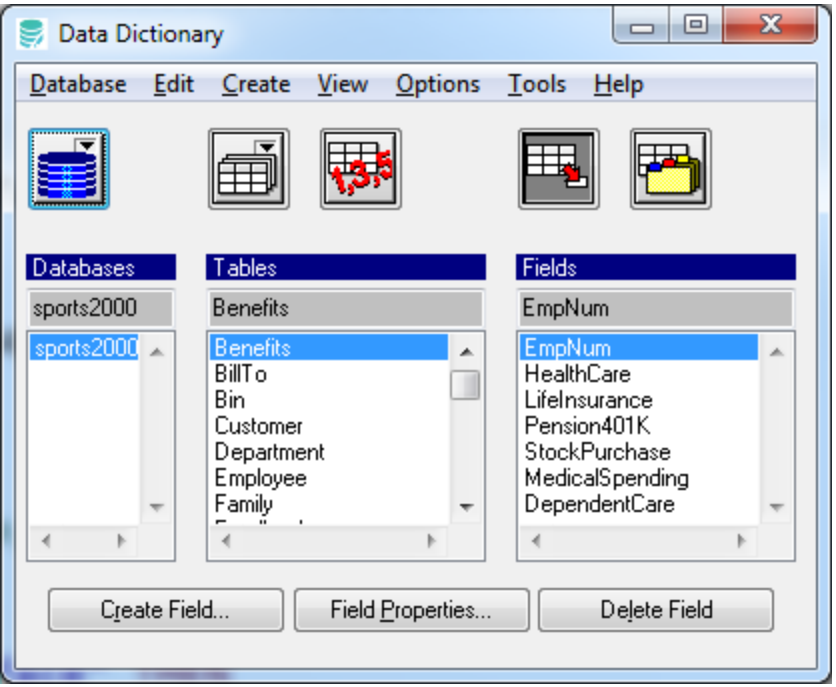


Everything goes through the DA layer

Building and deploying a DA layer is a lot like the game of "GO"

How do we get started?

Code Generation Requirements



For each table in the current database:

- Create a TT definition include file.
- Create a dataset definition include file.
- Create a data-source include file.

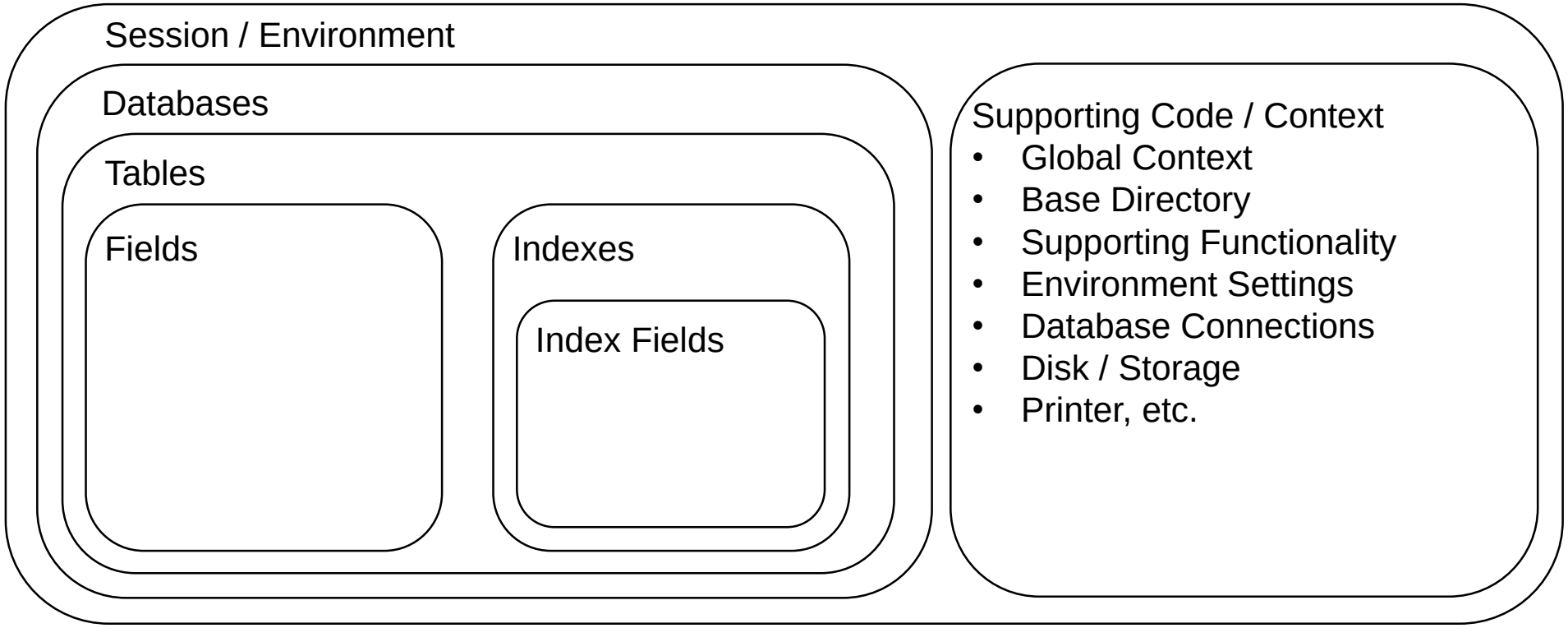
Also create the following classes:

- Create a domain class with properties that are a 1:1 map to each field in the table
- Create a TT/PDS access class with properties that are a 1:1 map to each field in the table and routes all reads/writes to the corresponding TT field
- Create a DB table access class with properties that are a 1:1 map to each field in the table and routes all reads/writes to the corresponding DB table field
- Create an interface class with properties that are a 1:1 map to each field in the table
- Have all classes implement the corresponding table's interface class.

See the handouts for some examples

Modeling The Environment

Visualize the application environment using encapsulation:

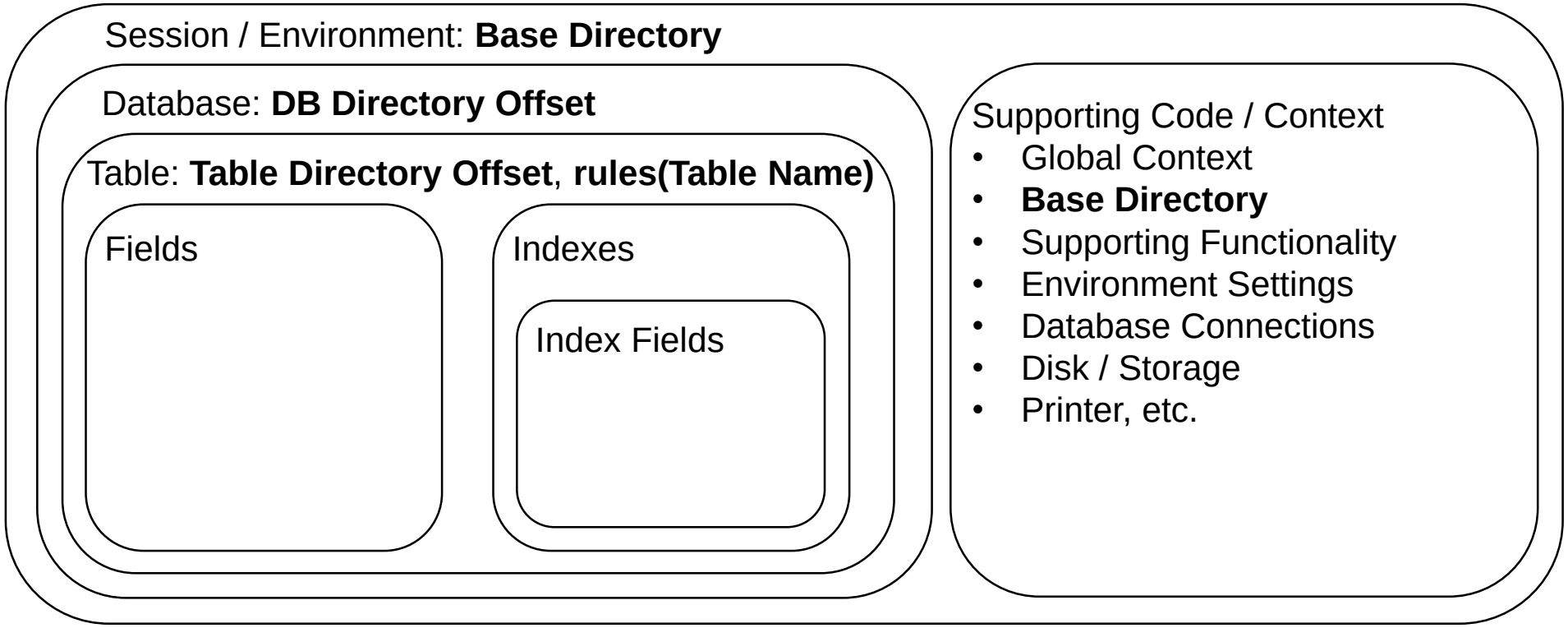


- Session / Environment – global context, base directory, overall application configuration
- Databases – collection of databases
- Tables – collection of tables in a database
- Fields – collection of fields in a table
- Indexes – collection of indexes for a table
- Index Fields – collection of table fields which define an index

How does this look like in actual use?

Establishing the Storage Structure

Visualize the application environment using encapsulation:



Output File Elements:

- Session / Environment - Base Directory
- Database - DB Directory Offset
- Table - Table Directory Offset
- File Name - rules(Table Name, output file type)

File Naming Rules

File Naming Rules:

Name	Structure
Interface Class	"I" + table name + ".cls"
Domain Class	table name + ".cls"
DB Access Class	"db" + table name + ".cls"
Dataset Access Class	"pds" + table name + ".cls"
TT Definition File	"tt" + table name + ".tt"
ProDataSet Definition File	"pds" + table name + ".pds"
Data Source Definition File	"dsrc" + table name + ".i"
&GLOBAL-DEFINE Sentinel	*Offset file path* + "__" Directory and file separators are replaced with "-"

What structure can we infer from these rules?

1. Names are composed of *prefix* + *table name* + *suffix* + "." + *extension*
2. Sentinel names use the offset file path and replace the directory/file separators with "-"
3. Because this functionality is used for different output files, it's considered global functionality.

Paths and Names

Information needed to generate code files for each db table:

Base Directory:	D:\Code\DataAccess
DB Directory Offset:	Sports2000
DB Directory Path:	D:\Code\DataAccess\sports2000
Table Directory Offset:	Benefits
DB Table Directory Offset:	Sports2000\Benefits

Base Directory: D:\Code\DataAccess

DB Directory Offset: Sports2000
DB Directory Path: D:\Code\DataAccess\sports2000

Table Directory Offset: Benefits

DB Table Directory Offset: Sports2000\Benefits
DB Table Directory Path: D:\Code\DataAccess\sports2000\Benefits
DB Table File Path: D:\Code\DataAccess\sports2000\Benefits\Benefits.cls

Q: Where would you expect to see these names in the code base?

Supporting Code: FileSystem.cls

Need ability to manipulate the filesystem:

- Ensure a given directory path has a standard directory separator
- Create directory path
- Combine list of directory path components into a fully qualified directory path
- Delete a directory path
- Check if a path exists
- Check if a path points to a directory

Filesystem Utilities: Core/FileSys/FileSystem.cls

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

CONSTRUCTOR                 FileSystem():
CONSTRUCTOR                 FileSystem(chDirPathSeperator   AS CHARACTER):

METHOD PUBLIC CHARACTER     GetStandardDirectoryFilePath(chPath AS CHARACTER):
METHOD PUBLIC LOGICAL       CreateDirectoryPath(chTargetDirectoryPath AS CHARACTER):
METHOD PUBLIC LOGICAL       DeleteDirectoryPath(chTargetDirectoryPath AS CHARACTER):
METHOD PUBLIC CHARACTER     CombineDirectoryPathList(chPathList AS CHARACTER):
METHOD PUBLIC LOGICAL       isDirectory(chDirectoryPath  AS CHARACTER):
METHOD PUBLIC LOGICAL       CheckPathExists(chTargetPath AS CHARACTER):
```

Supporting Code: Immutable Value Object

When generating code for a DB table, there is a need to hold the table configuration values in a place that is effectively "read-only".

Immutable Value Object:

Immutable:
Unchanging over time / unable to be changed.

Value Object:
Object containing a collection of values which are considered equal if all their values are equal.

Immutable + Value object = "write all values in an object once, read only after."

Supporting Code: GenerateSupport


```
USING Core.Filesys.FileSystem           FROM PROPATH.

CLASS Generate.Code.GenerateSupport
    INHERITS FileSystem:

DEFINE PUBLIC PROPERTY BaseDirectory        AS CHARACTER    NO-UNDO GET. PRIVATE SET.
DEFINE PUBLIC PROPERTY DirectoryOffset      AS CHARACTER    NO-UNDO GET. PRIVATE SET.
DEFINE PUBLIC PROPERTY FullDirectoryPath    AS CHARACTER    NO-UNDO GET. PRIVATE SET.
DEFINE PUBLIC PROPERTY TabSize              AS INTEGER      NO-UNDO GET. PRIVATE SET.

CONSTRUCTOR GenerateSupport(chBaseDir   AS CHARACTER,
                            chDirOffset AS CHARACTER,
                            iTabSize    AS INTEGER
                            ):
SUPER().        /* Make sure the base class is setup first  */

ASSIGN
    THIS-OBJECT:BaseDirectory       = THIS-OBJECT:GetStandardDirectoryFilePath(chBaseDir)
    THIS-OBJECT:DirectoryOffset     = THIS-OBJECT:GetStandardDirectoryFilePath(chDirOffset)
    THIS-OBJECT:FullDirectoryPath   = THIS-OBJECT:CombineDirectoryPathList(
                                            THIS-OBJECT:BaseDirectory + "," +
                                            THIS-OBJECT:DirectoryOffset)
    THIS-OBJECT:TabSize             = iTabSize
    .

END CONSTRUCTOR.
END CLASS.
```

Inherit filesystem support class

Property only set in constructor, can be read anywhere

GenerateSupport Discussion

GenerateSupport inherits from the FileSystem class

Questions:

- Directory Specifications - can be argued they fit with the FileSystem class
- DirectoryOffset property - could be renamed DbTableOffset
- TabSize - not a filesystem related value – does it belong here?
- Is there a more accurate name that describes what GenerateSupport does?
- For the GenerateSupport and FileSystem classes, what are the implications of using inheritance vs composition?

OO - Figuring Out "What goes Where"

OO development "off the cuff"
vs
up-front planning

Identifying Data Source Definition Elements

Identify elements in the code below:

```
&IF DEFINED(sports2000-Benefits-dsrcBenefits-i__) = 0 &THEN
&GLOBAL-DEFINE sports2000-Benefits-dsrcBenefits-i__

DEFINE DATA-SOURCE dsrcBenefits FOR Benefits.
&ENDIF
```

Answer:

Database Name:	sports2000
Table Name:	Benefits
DataSource Name:	"dsrc" + Table Name
File Name:	"dsrc" + Table Name + ".i"
File Extension:	i
DB Table File Path:	sports2000/Benefits/dsrcBenefits.i
Wrapper Header:	&IF … &THEN
Wrapper Trailer:	&ENDIF
Wrapper &GLOBAL-DEFINE:	Derived from the *DB Table File Path*

Identifying Temp Table Definition Elements

Identify elements in the code below:

```
&IF DEFINED(sports2000-Benefits-ttBenefits-tt__) = 0 &THEN
&GLOBAL-DEFINE sports2000-Benefits-ttBenefits-tt__

DEFINE TEMP-TABLE ttBenefits NO-UNDO
    BEFORE-TABLE btBenefits
    FIELD EmpNum                        AS INTEGER     INITIAL 0
                                                       FORMAT "zzzzzzzzz9"    LABEL "Emp No"
    FIELD HealthCare                    AS CHARACTER   FORMAT "x(8)"          LABEL "Health Care"
    FIELD LifeInsurance                 AS INTEGER     INITIAL 0
                                                       FORMAT "$>>,>>>,>>>"   LABEL "Life Insurance"

    INDEX EmpNo                 IS PRIMARY UNIQUE  EmpNum
    .

&ENDIF
```

Padded Attribute

Padded Quoted Attribute

Answer:
Same as the DataSource slide, plus…

TempTable Name:	"tt" + TableName
BeforeTable Name:	"bt" + TableName
Field Name:	FOR EACH _file for a given TableName …
Padded Attribute:	Strings space padded to a certain length – Field, Index names, language elements
Padded Quoted Attribute:	Quoted strings space padded to a certain length – Label, Format

Identifying ProDataSet Definition Elements

Identify elements in the code below:

```
&IF DEFINED(sports2000-Benefits-pdsBenefits-pds__) = 0 &THEN
&GLOBAL-DEFINE sports2000-Benefits-pdsBenefits-pds__

{sports2000\Benefits\ttBenefits.tt}

DEFINE DATASET dsBenefits
    FOR ttBenefits.

&ENDIF
```

Answer:

Same as the DataSource slide, plus…

ProDataSet Name: "ds" + Table Name
TT Name: "tt" + Table Name

TT include: "{" + DB Table Offset + "tt" + Table Name + ".tt" + "}"

Identifying Domain Object Elements

Identify elements in the code below:

```
CLASS sports2000.Benefits.Benefits
    IMPLEMENTS sports2000.Benefits.IBenefits
    :

DEFINE PUBLIC PROPERTY EmpNum              AS INTEGER      NO-UNDO GET. SET.
DEFINE PUBLIC PROPERTY HealthCare          AS CHARACTER    NO-UNDO GET. SET.
DEFINE PUBLIC PROPERTY LifeInsurance       AS INTEGER      NO-UNDO GET. SET.

END CLASS.
```

Answer:

Same as the DataSource slide, except …

Wrapper name:	CLASS
Class package name:	rules(DB Table File Path)
Interface package name:	rules(DB Table File Path)

Same as TT Definition File:

Padded Attribute:	Property Names, Data type

Also added:

Property Attribute:	GETter and SETer code

Identifying DB Access Object Elements

Identify elements in the code below:

```
CLASS sports2000.Benefits.dbBenefits
    IMPLEMENTS sports2000.Benefits.IBenefits:

DEFINE PUBLIC PROPERTY EmpNum    AS INTEGER          NO-UNDO
                                 GET():
                                 RETURN(Benefits.EmpNum).
                                 END GET.
                                 SET(tmpvar AS INTEGER):
                                 ASSIGN Benefits.EmpNum = tmpvar.
                                 END SET.

END CLASS.
```

Answer:
Same as the DomainObject except …

Property Attribute: GETter and SETer code maps to a TT table buffer

Identifying PDS Access Object Elements

Identify elements in the code below:

```
CLASS sports2000.Benefits.pdsBenefits
    IMPLEMENTS sports2000.Benefits.IBenefits:

{sports2000\Benefits\pdsBenefits.pds}

DEFINE PUBLIC PROPERTY EmpNum    AS INTEGER         NO-UNDO
                                 GET():
                                 RETURN(ttBenefits.EmpNum).
                                 END GET.
                                 SET(tmpvar AS INTEGER):
                                 ASSIGN ttBenefits.EmpNum = tmpvar.
                                 END SET.
END CLASS.
```

Answer:
Same as the DomainObject except …

TT Definition Include: "{" + rules(DB Table Path) + "}"
Property Attribute: GETter and SETer code maps to a TT table buffer

Identifying Interface Object Elements

Identify elements in the code below:

```
INTERFACE sports2000.Benefits.IBenefits:

DEFINE PUBLIC PROPERTY EmpNum           AS INTEGER      NO-UNDO  GET. SET.
DEFINE PUBLIC PROPERTY HealthCare       AS CHARACTER    NO-UNDO  GET. SET.
DEFINE PUBLIC PROPERTY LifeInsurance    AS INTEGER      NO-UNDO  GET. SET.

END INTERFACE.
```

Answer:
Same as the DomainObject except …

Interface:	Not included
Wrapper name:	INTERFACE

Bringing It All Together

OO Concepts:

- Separation of Concerns
- Inheritance – More Specific to More General
- Using common structures across classes to facilitate refactoring and re-use

Candidates for Inheritance:

- Same as the DataSource slide, plus…
- Same as the DomainObject …

Categorizing Application Classes

Definition Generation Files

- GenerateDataSource.cls
- GenerateDomainClass.cls
- GenerateTTDefinition.cls
- GeneratePDS.cls

Access Generation Files

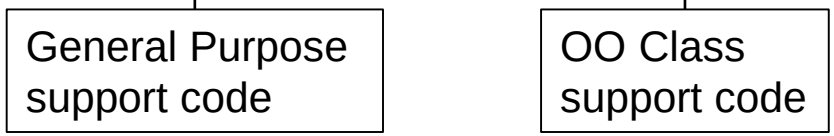
- GenerateDbClass.cls
- GenerateTTClass.cls
- GenerateInterfaceClass.cls

General Purpose Support Classes

- GenerateIndirectionClassBase.cls
- GenerateBase.cls

Global Support Classes

- GenerateSupport.cls
- FileSystem.cls



Class Structures as Implemented

Sub Class
Super Class

GenerateDataSource.cls
GenerateBase.cls

GenerateDbClass.cls
GenerateIndirectionClassBase.cls
GenerateBase.cls

GenerateDomainClass.cls
GenerateBase.cls

GenerateTTClass.cls
GenerateIndirectionClassBase.cls
GenerateBase.cls

GeneratePDS.cls
GenerateBase.cls

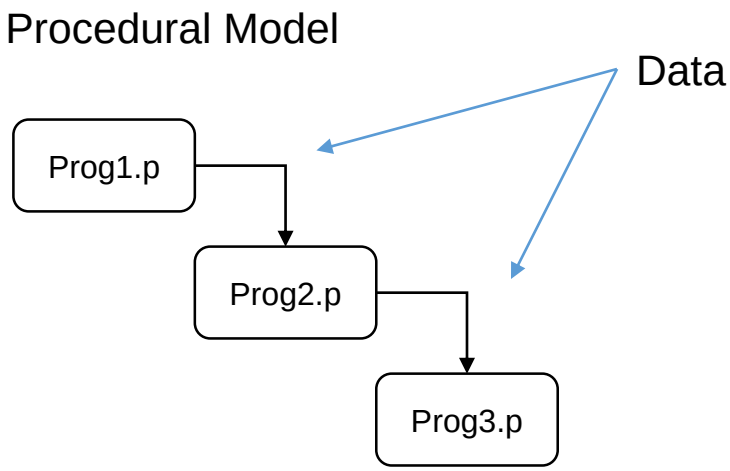
GenerateInterfaceClass.cls
GenerateIndirectionClassBase.cls
GenerateBase.cls

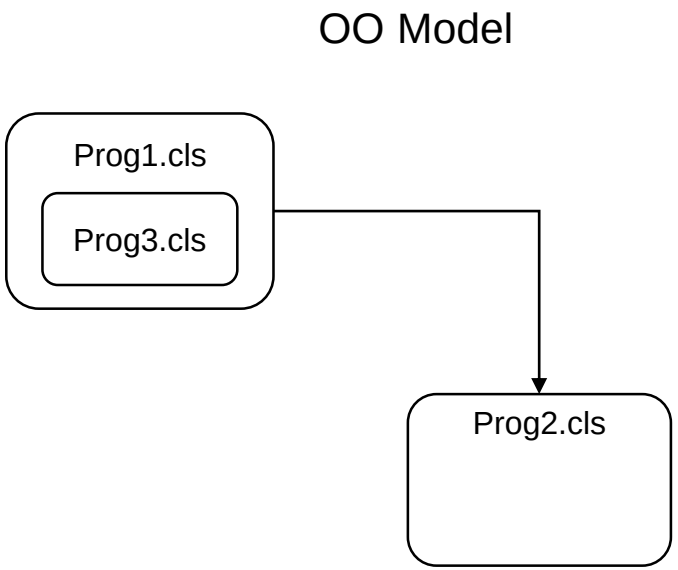
GenerateTTDefinition.cls
GenerateBase.cls

GenerateSupport.cls
FileSystem.cls

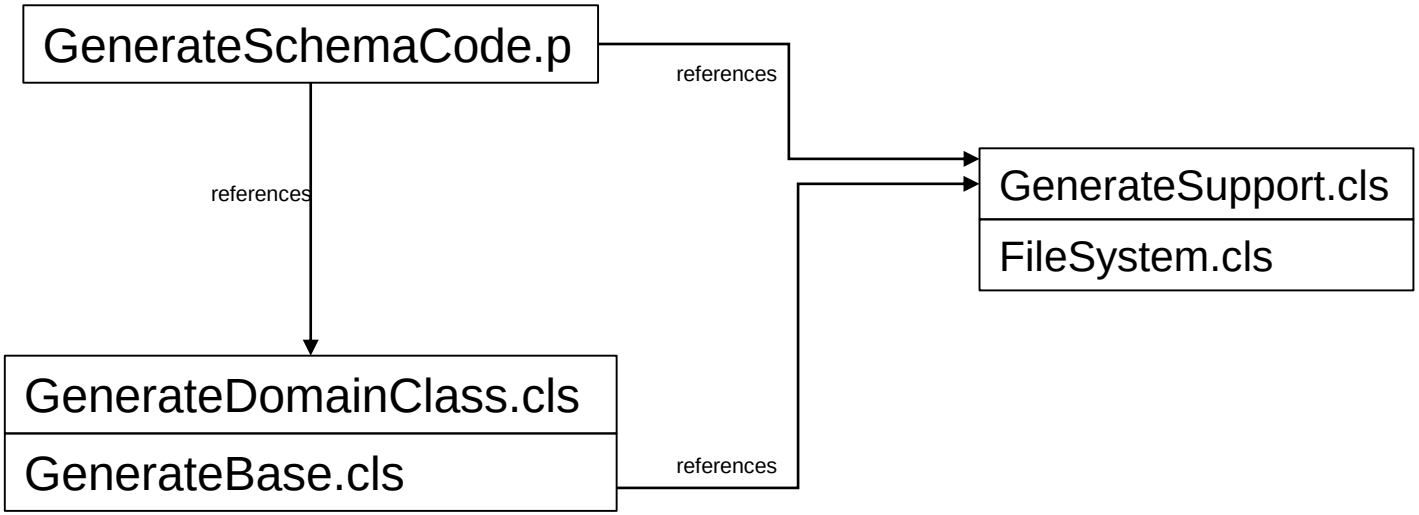
Dependency Injection

What does Dependency Injection look like?





Dependency Injection Overview



One instance of GenerateSupport,
multiple references to the object instance

Dependency Injection in Action


```
ASSIGN
    oGenerateSupport  = NEW GenerateSupport(chBaseDirectory, chDbDirOffset, 4).
```

"Child" class instantiated

then passed to a subordinate class

```
oGenerateDomainClass    = NEW GenerateDomainClass(  oGenerateSupport,
                                                    DbTables._file-name
                                                    ).
```

Dependency Injection in Action


```
CLASS Generate.Code.GenerateDomainClass
    INHERITS GenerateBase:

/*************************************************************************/
CONSTRUCTOR GenerateDomainClass(oGenSupport  AS GenerateSupport,
                                chTableName  AS CHARACTER):
/*************************************************************************/
SUPER(  oGenSupport,
        chTableName,
        "CLASS"
        ).



END CONSTRUCTOR.
```

Super Class

Instance reference received

Instance reference passed to super class

Dependency Injection in Action

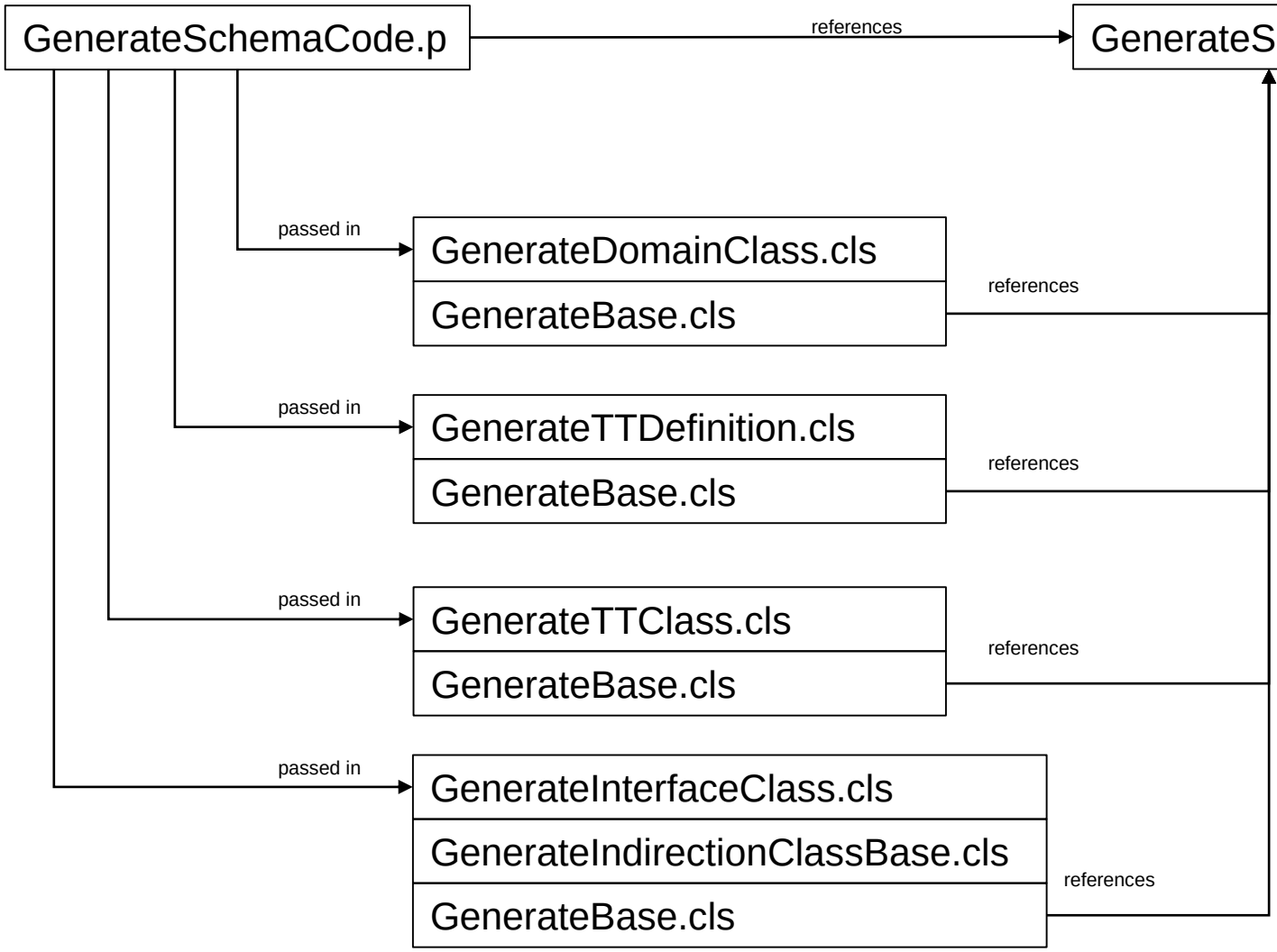

```
CLASS Generate.Code.GenerateBase:

/*************************************************************************/
CONSTRUCTOR GenerateBase(oGenSupport     AS GenerateSupport,
                         chTableName     AS CHARACTER,
                         chFileType      AS CHARACTER):
/*************************************************************************/
/* Initialize                                */

ASSIGN
    THIS-OBJECT:GenerateSupport = oGenSupport
    THIS-OBJECT:TableName       = chTableName
    THIS-OBJECT:FileType        = chFileType

    THIS-OBJECT:FullTablePath   = THIS-OBJECT:GetFullTablePath()

    THIS-OBJECT:OffsetTablePath = THIS-OBJECT:GenerateSupport:CombineDirectoryPathList(
                                  THIS-OBJECT:GenerateSupport:DirectoryOffset + "," +
                                  THIS-OBJECT:TableName
                                  ).

END CONSTRUCTOR.
```

Instance reference received

Save properties common to all generation classes

Use injected functionality

Dependency Injection – Final Arrangement



Single instance used in multiple parts of the system provides shared context while controlling scope

Wrapper / Body Implementation

Each generated file has a header, body, and footer section.

Since this behavior is common to all generated files, we could code a base class like so:

```
METHOD PROTECTED VOID GenerateHeader():
METHOD PROTECTED VOID Generate Body():
METHOD PROTECTED VOID GenerateFooter():
```

This provides the required functionality, and allows it to be overridden when need be.

In the current code base, the Generate*Body*() functionality is implemented in different ways.

GenerateIndirectionClassBase.cls


```
METHOD PROTECTED VOID GenerateClassFile():

/* Write this to a .cls file                        */
OUTPUT
    TO VALUE(THIS-OBJECT:GetFullTablePathFileName(THIS-OBJECT:ClassNamePrefix, "", "cls")).

/* Generate the top part of the class file          */
THIS-OBJECT:GenerateHeader().

/* If this table has fields then generate properties for them   */
/* If not, then generate a 'no fields here' comment             */
IF THIS-OBJECT:CheckIfThisTableHasAnyFields(RECID(DbTable)) THEN
    THIS-OBJECT:GenerateProperties().
ELSE
    PUT UNFORMATTED
        THIS-OBJECT:GenerateCommentTableHasNoFields()   SKIP.

/* Generate the ending part of the class            */
THIS-OBJECT:GenerateFooter().

OUTPUT CLOSE.

END METHOD.
```

Generate Header Code

Generate Body Code

Generate Footer Code

Overriding the GenerateHeader Method

When the default header isn't what we want for this class – override it!

GenerateIndirectionClassBase.cls

Base Class

```
METHOD PROTECTED VOID GenerateHeader():
SUPER:GenerateClassHeader(THIS-OBJECT:ClassNamePrefix, "", "").
END METHOD.
```

GenerateTTClass.cls

SubClass that inherits GenerateIndirectionClassBase.cls

```
METHOD PROTECTED OVERRIDE VOID GenerateHeader():

/* Generate the CLASS header text                */
SUPER:GenerateHeader().

/* Generate the dataset include file reference  */
PUT UNFORMATTED
    "~{" + THIS-OBJECT:GetOffsetTablePathFileName(THIS-OBJECT:ClassNamePrefix,
                                                  "", "pds") + "~}"
    SKIP(1).

END METHOD.
```

Overriding the GetPropertyGetSet Method

When the default Get/Set code isn't what we want…

GenerateIndirectionClassBase.cls

Base Class

```
METHOD PROTECTED CHARACTER GetPropertyGetSet():
DEFINE VARIABLE chRetval AS CHARACTER NO-UNDO.
ASSIGN
    chRetval =                                          CHR(10) +
               FILL(" ", 75) + "GET(&1):"             + CHR(10) +
               FILL(" ", 75) + "RETURN(&2)."          + CHR(10) +
               FILL(" ", 75) + "END GET."             + CHR(10) +
               FILL(" ", 75) + "SET(&3):"             + CHR(10) +
               FILL(" ", 75) + "ASSIGN &2 = &4."      + CHR(10) +
               FILL(" ", 75) + "END SET.".
RETURN(chRetval).
END METHOD.
```

GenerateInterfaceClass.cls

SubClass that inherits GenerateIndirectionClassBase.cls

```
METHOD PROTECTED OVERRIDE CHARACTER GetPropertyGetSet():
DEFINE VARIABLE chRetval AS CHARACTER NO-UNDO.

ASSIGN
    chRetval =  " GET. SET.".

RETURN(chRetval).
END METHOD.
```

Q&A?

Thank You For Your Time!