



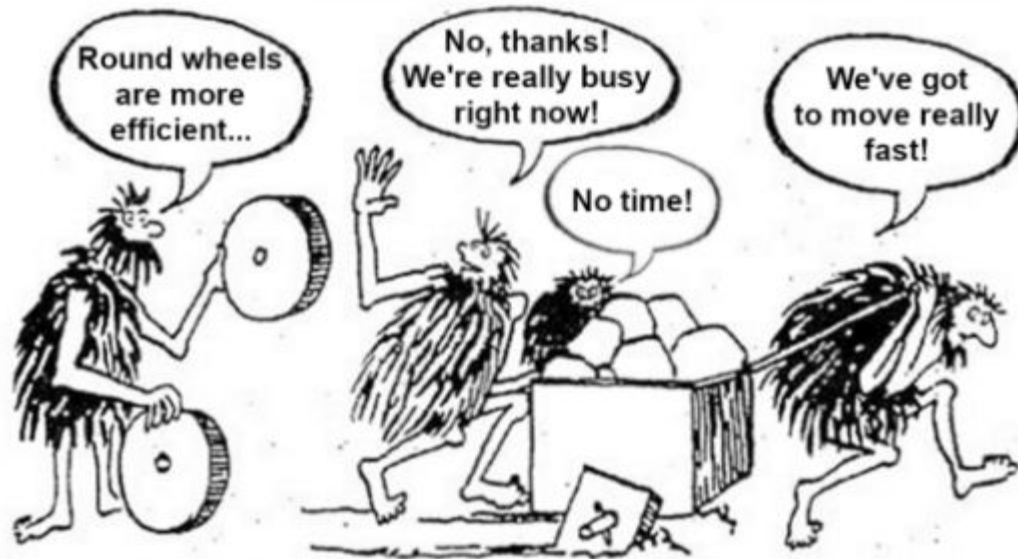
Refactoring AB Suite Applications to Reduce Maintenance Effort

A byproduct of Speed to Market

Refactoring – What is it, Why Bother?

- According to Wikipedia, Refactoring is ...
 - The process of restructuring existing computer code without changing its external behaviour
 - Intended to improve the design, structure, and/or implementation of the software (its non-functional attributes), while preserving its functionality
- Potential advantages of refactoring may include ...
 - Improved code readability
 - Reduced code complexity
 - Improving the source code's maintainability
 - Creating a simpler, cleaner, object model to improve extensibility
 - Improved testing capabilities

Refactoring – Typical Objections





Refactoring Approaches

Improve Code Clarity

- Genuine Constants

- Any attribute (variable) can be defined with an Initial Value, like EAE
- But now you can mark the variable as “IsConstant”, and the Editor will syntax any attempts in LDL+ to assign a value to the variable
- A simple, and easy to implement, improvement to code quality and clarity

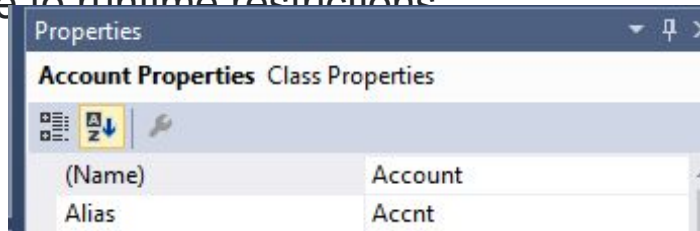
Move 25 DMVDefaultRecSize

(attribute) Signed Number (Length: 11) DMVDefaultRecSize

Operand DMVDefaultRecSize is Read-only (Property IsConstant = True)

- Meaningful Names

- The EAE name limitations no longer apply
- Elements have an Alias property that is used to adhere to runtime restrictions
 - e.g. Database schema elements, Ispecs, Reserved Words
- Unlike EAE it's easy to rename elements in AB Suite
 - Makes EAE-style name abbreviation standards redundant



Properties	
Account Properties Class Properties	
(Name)	Account
Alias	Accnt

Make More Use of Methods

- Segment-level methods may be marked as “Public”, allowing them to be

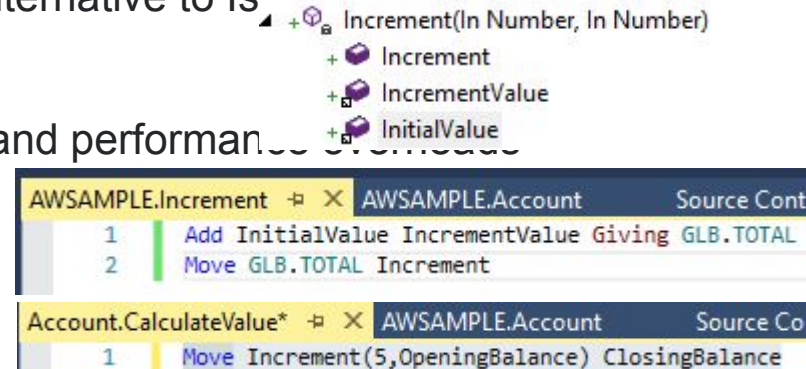
- Invoked via Client Tools and ePortal as an alternative to Ispace
- Called from external programs
- Don't carry the Framework cycle behaviour and performance overhead

- Methods can be parameterised

- With multiple parameters of different types
 - e.g. String, Number, Boolean, Array
- Reduce/eliminate need for GSDs over time

- Methods can return a result

- Similar to typed procedures/functions in other languages



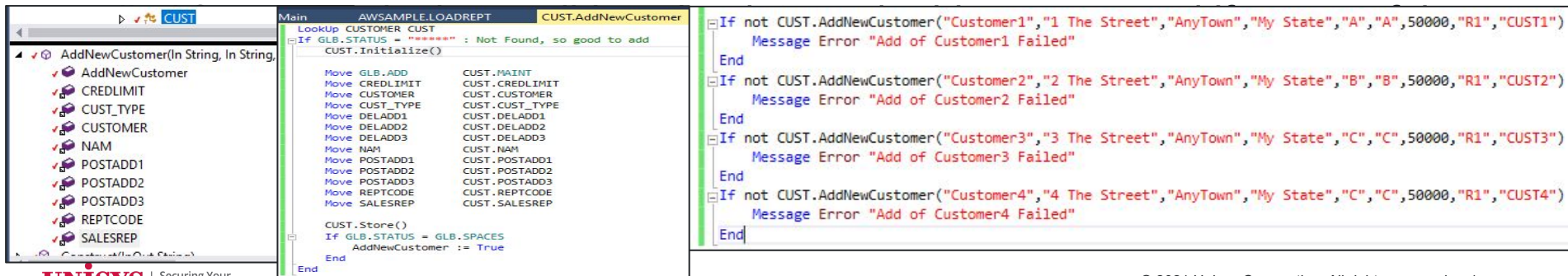
The screenshot displays a software interface with two panels. The top panel shows a list of methods: 'Increment(In Number, In Number)', 'Increment', 'IncrementValue', and 'InitialValue'. The bottom panel shows a table with two sections. The first section is titled 'AWSAMPLE.Increment' and contains two rows: '1 Add InitialValue IncrementValue Giving GLB.TOTAL' and '2 Move GLB.TOTAL Increment'. The second section is titled 'Account.CalculateValue*' and contains one row: '1 Move Increment(5,OpeningBalance) ClosingBalance'.

AWSAMPLE.Increment		AWSAMPLE.Account	Source Cont
1	Add InitialValue IncrementValue Giving GLB.TOTAL		
2	Move GLB.TOTAL Increment		

Account.CalculateValue*		AWSAMPLE.Account	Source Co
1	Move Increment(5,OpeningBalance) ClosingBalance		

Make More Use of Methods

- User-defined Methods can be created at the Segment level, in Ispecs, and in Reports
 - Report methods for MCP Runtime require AB Suite 8.0
 - You're no longer limited to just the Construct, Prepare, and Main framework methods
- Great way to minimise code duplication and complexity
- For example, the CUST Ispec has an "AddNewCustomer" method that can be



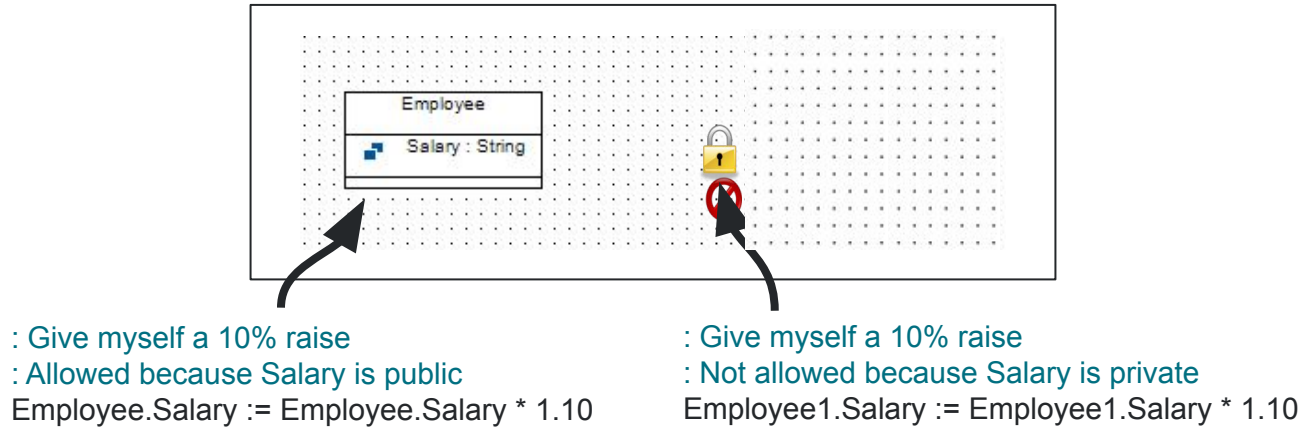
Encapsulate Wherever Possible

- Encapsulation is one of the core concepts of Object-Oriented programming
- Encapsulation is the implementation of self-contained code modules that can be modified and tested in isolation – properly implemented they ...
 - Run successfully using only data passed in and out as parameters and return values
 - Can reduce/eliminate the need for large scale application regression testing
 - Enable the use of test harnesses
- AB Suite enables encapsulation by using methods with
 - Parameters
 - Return values
 - Restricted data visibility

Enforce Business Rules for Persistent Data

- Any ex-EAE application is like the “wild west” – lawless
 - Any persistent data item in any Ispec class can be updated by code anywhere in the application
 - Organisations often use design and development standards to try prevent this
 - But there’s no inbuilt enforcement of the standards
- Use the Visibility property of an attribute to turn on enforcement within the model
 - Set Visibility to Private to restrict access just to code written within the owning Ispec class
 - Any existing code outside of the owning Ispec class will be invalidated by this change
 - Add new Public methods to the Ispec class to provide the necessary functionality
 - Code in these new methods can access all attributes within their owning Ispec class
 - Code outside of the Ispec class can only access the persistent attributes via these new methods

Enforce Business Rules for Persistent Data



Use the Model to Implement Enforcement

SREP.SALESREP x AWSAMPLE.SREP Source Control Explorer

Depends on

Depended on by

Source	Kind	Target	Status
CUST.Construct	Use	SALESREP	Valid
LOADREPT.Main	Use	SALESREP	Valid
SALESREPEX.Frame01....	Use	SALESREP	Valid
SALESREPLD.Main	Use	SALESREP	Valid
SERCH.Main	Use	SALESREP	Valid
SREP	Use	SALESREP	Valid
SREPBYAREA.Frame10...	Use	SALESREP	Valid

Depended on by

Source	Kind	Target	Status
CUST.Construct	Use	SALESREP	Needs Validation
LOADREPT.Main	Use	SALESREP	Needs Validation
SALESREPEX.Frame01....	Use	SALESREP	Needs Validation
SALESREPLD.Main	Use	SALESREP	Needs Validation
SERCH.Main	Use	SALESREP	Needs Validation
SREP	Use	SALESREP	Needs Validation
SREPBYAREA.Frame10...	Use	SALESREP	Needs Validation
SREPBYAREA.Main	Use	SALESREP	Needs Validation

Properties

SALESREP Properties Attribute Properties

(Name)	SALESREP
AlphaClearWhenCharacter	<none>
Author	Wardle, Andy
Caption	Sales Rep Number
Value	
VersionFile	
Visibility	Private

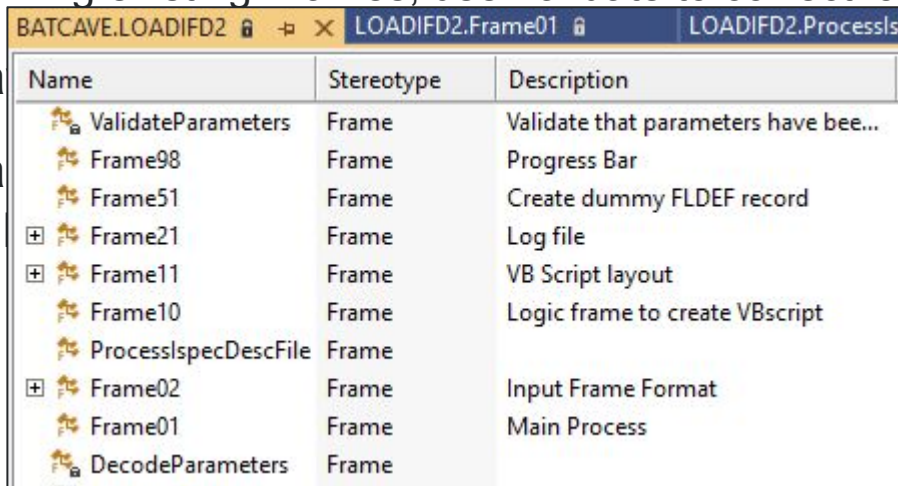
CUST.Construct x SREP.SALESREP AWSAMPLE.SREP Source Control Explorer

```
1 Move GLB.SPACES SD_RECORD.SD_SREP
2 Move GLB.SPACES SD_RECORD.SD_REPNAME
3 LookUp Every SREP
4   Move SREP.SALESREP SD_RECORD.SD_SREP
5   Move SREP.SALESREP SD_RECORD.SD_REPNAME
6   SendListDy
7 End
```

Operand 'SREP.SALESREP' is inaccessible due to its Visibility setting

Improve Report Maintainability

- Reports can have more than 99 Frames
- Frames now have proper names, not simply a number
 - Use meaningful names for new Frames
 - Consider renaming existing Frames, use Validate to correct logic references



Name	Stereotype	Description
ValidateParameters	Frame	Validate that parameters have bee...
Frame98	Frame	Progress Bar
Frame51	Frame	Create dummy FLDEF record
Frame21	Frame	Log file
Frame11	Frame	VB Script layout
Frame10	Frame	Logic frame to create VBscript
ProcessSpecDescFile	Frame	
Frame02	Frame	Input Frame Format
Frame01	Frame	Main Process
DecodeParameters	Frame	

- Reports can have more than 99 Frames (e.g. 8.0 for MCP)
- More Frames allow for the ability to decompose large blocks of code
- Improves code readability by reducing complexity and improving

Wrap Existing Methods

- Some methods may appear to be difficult to refactor, but the refactoring them offers potential benefits
- One solution is to “wrap” the existing method inside a new method
 - The new method would use parameters and a return value, and use them to set up the input and output segment attributes (EAE GSDs) that the existing method requires
 - Any new logic would use the new method
 - Over time any logic references to the existing method can be replaced with the new method, until none remain
 - At which point the existing method can be seamlessly refactored or replaced
- This would improve code maintainability and reduce complexity
 - If the new method can be encapsulated that would be an additional benefit

Wrap Existing Methods

```
AWSAMPLE.SecurityCheck  X MENU.Prepare
1  Move GLB.SPACES GD_Error
2  LookUp GD_User SREP
3  If GLB.STATUS = "*****"
4  |   Move "*" GD_Error
5  |   Else
6  |       IsUserAllowedToUseISPEC()
7  |   End
```

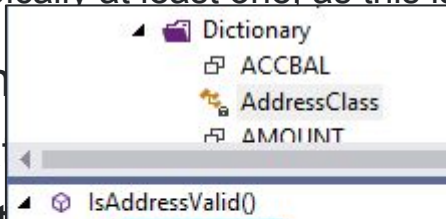
```
MENU.Prepare  X AWSAMPLE.SecurityCheck  AWSAMPLE.SecurityCheck
1  Runtime_Context__ := "ISPEC" : Added by migration
2  Insert ACTION_LINE ( )
3
4  Move ISPEC          GD_ISPEC
5  Move GLB.USERCODE   GD_User
6  SecurityCheck()
7  If GD_Error = "*"
8  |   Message Error "Invalid User"
9  |   Else
10 |       If GD_Error = "Y"
11 |           Message Error "Invalid User/ISPEC"
12 |       End
13 |   End
14 |   If GD_Error <> GLB.SPACES
15 |       EndExit
```

```
AWSAMPLE.SecurityCheckWrapper  X MENU.Prepare  AWSA
1  GD_ISPEC := MyISPEC
2  GD_User := MyUser
3  SecurityCheck()
4  If GD_Error = "*"
5  |   Message Error "Invalid User"
6  |   Else
7  |       If GD_Error = "Y"
8  |           Message Error "Invalid User/ISPEC"
9  |       End
10 |   End
11 |   If GD_Error <> GLB.SPACES
12 |       Return True
13 |   Else
14 |       Return False
15 |   End
```

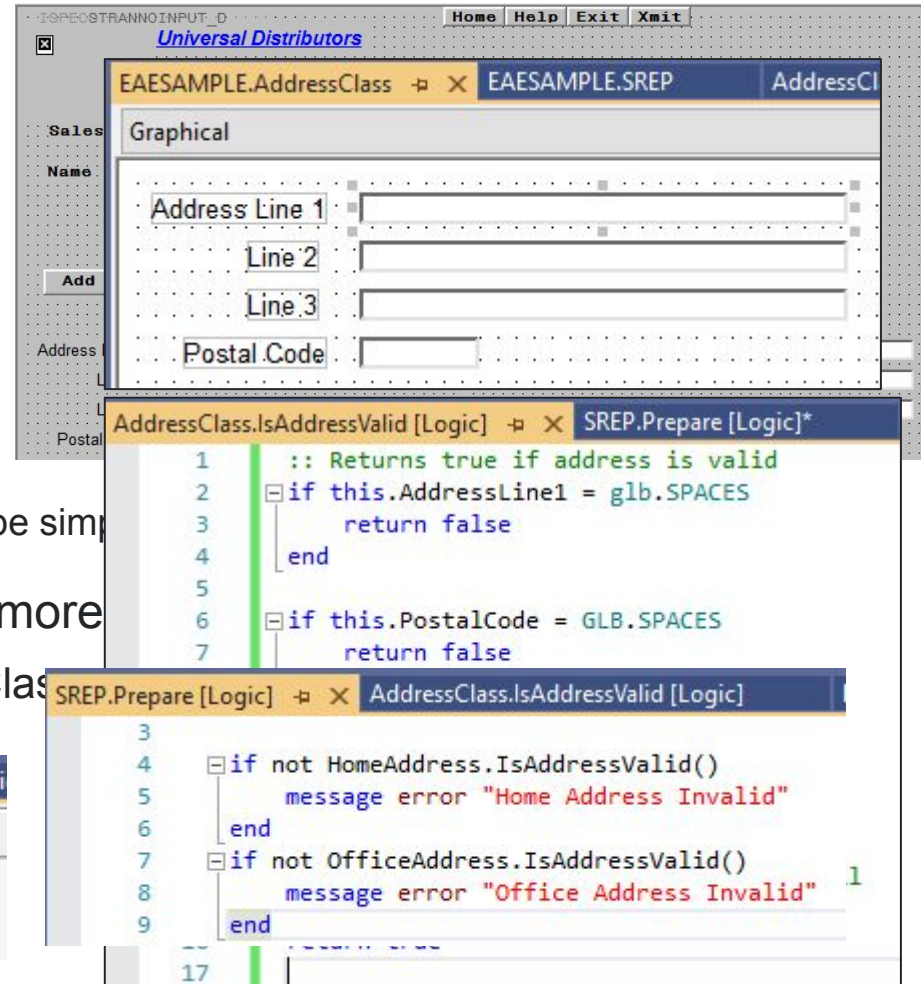
```
MENU.Prepare  X AWSAMPLE.SecurityCheck  AWSAMPLE.SecurityCheck
1  Runtime_Context__ := "ISPEC" : Added by migration
2  Insert ACTION_LINE ( )
3
4  If Not SecurityCheckWrapper(ISPEC, GLB.USERCODE)
5  |   : We had an error so exit now
6  |   EndExit
```

Use Vanilla Classes

- These are complex Dictionary Items with
 - One or more attributes
 - Zero or more painted items
 - Zero or more methods
- Typically at least one, as this is how code can be simplified
- The whole idea is to look one or more methods up in the Vanilla Class Dictionary and use the Vanilla Class IsAddressValid() method used was
- Not



Name	Kind	Stereotype	Template
HomeAddress	Attribute		EAESAMPLE.AddressClass
OfficeAddress	Attribute		EAESAMPLE.AddressClass
ACTION LINE	Attribute		FAFSAMPLE.ACTION LINE



Use Multiple Ispec Instances

- Multiple instances of an Ispec can be defined and referenced in logic
- Each instance is a differently named copy of the Ispec
 - The original Ispec is the template for each copy
 - Template updates are automatically reflected in each instance
 - Minimal amount of logic is required to manage each instance
- A typical use case might be a bank account with multiple account holders
- Allows the data for each instance to be held in memory concurrently

Use Multiple Ispec Instances

- Simple example using multiple instances of an Ispec
 - Customer A is merged with Customer B to create Customer C.

```
LookUp CUSTOMERA (CUST)
Move CUST.CUSTOMER      SD_CUSTA_CUSTOMER
Move CUST.NAM           SD_CUSTA_NAM
Move CUST.POSTADD1      SD_CUSTA_ADD1
Move CUST.POSTADD2      SD_CUSTA_ADD2
Move CUST.POSTADD3      SD_CUSTA_ADD3
Move CUST.CREDLIMIT     SD_CUSTA_CRLIM
```

```
LookUp CUSTOMERB (CUST)
Move CUST.CUSTOMER      SD_CUSTB_CUSTOMER
Move CUST.NAM           SD_CUSTB_NAM
Move CUST.POSTADD1      SD_CUSTB_ADD1
Move CUST.POSTADD2      SD_CUSTB_ADD2
Move CUST.POSTADD3      SD_CUSTB_ADD3
Move CUST.CREDLIMIT     SD_CUSTB_CRLIM
```

```
CUST.AutoIspec.ClearPersistent()
CUST.AutoIspec.Set_MAINT("ADD")
Move SD_MERGE_CUSTOMER CUST.AutoIspec.CUSTOMER
Move SD_CUSTA.NAM       CUST.AutoIspec.NAM
Move SD_CUSTA.ADD1      CUST.AutoIspec.POSTADD1
Move SD_CUSTA.ADD2      CUST.AutoIspec.POSTADD2
Move SD_CUSTA.ADD3      CUST.AutoIspec.POSTADD3
Add SD_CUSTA_CRLIM      SD_CUSTB_CRLIM Giving GLB.TOTAL
Move GLB.TOTAL          CUST.AutoIspec.CREDLIMIT
CUST := CUST.AutoIspec
CUST.Store()
```

Name	Kind	Template
CustomerA	Attribute	EAESAMPLE.CUSTOMER
CustomerB	Attribute	EAESAMPLE.CUSTOMER
MergedCustomer	Attribute	EAESAMPLE.CUSTOMER
CustAInstance	Attribute	EAESAMPLE.CUST
CustBInstance	Attribute	EAESAMPLE.CUST
MergedCustInstance	Attribute	EAESAMPLE.CUST

```
EAESAMPLE.CustMerge CustMerge.Main [Logic]*
1 lookup CustomerA CustAInstance
2 lookup CustomerB CustBInstance
3
4 MergedCustInstance.ClearPersistent()
5 :: Assign CustAInstance to MergedCustInstance instance
6 MergedCustInstance := CustAInstance
7 :: Now update fields with merged values
8 MergedCustInstance.CUSTOMER := MergedCustomer
9 MergedCustInstance.CREDLIMIT := CustAInstance.CREDLIMIT + CustBInstance.CREDLIMIT
10 MergedCustInstance.SetMaint("ADD")
11 MergedCustInstance.Store()
```

Use Complex Expressions

- Complex expressions in logic statements

Before:

```
Move CurrBal GD_CurrBal
Move IntrRate GD_IntrRate
GP_CalcInt()           : Result in GD_CalcRslt
Add 24 GD_Amount
Move 7 GD_Index
Compute ANumber (NumArray * GD_Amount * GD_CalcRslt)
```

After:

```
ANumber := NumberArray[7] * (Amount + 24) * CalculateInterest(CurrentBalance,InterestRate)
```

Before:

```
Multiply Salary Increment Giving GD_Inc
DoWhen (GD_Inc > MaxPayRiseAllowed)
    Move "Pay Rise Too Big = " GD_Msg
    AttachAndSpace GD_Inc GD_Msg
    Message Attention GD_Msg
End
```

After:

```
If (Salary * Increment) > MaxPayRiseAllowed
    Message Attention "Pay Rise Too Big =" &+ (Salary * Increment)
End
```

- Less code, fewer variables, and greater maintainability, for the same outcome

Use ForEach for Loops Wherever Possible

- ForEach – a new looping verb with two variants
- ForEach <Ispec Instance/Record> in <Ispec>/<Profile>/<Extract File>

- Alternative to looping Determine and Look In commands

```
ForEach CUST In CUST.CUSTNAMES
    Message Attention CUST.NAM &+ "is the Customer Name and" &+ CUST.CUSTOMER &+ "is the Customer ID"
End
```

- ForEach <Array Element> in <Array>
- Can iterate over arrays without needing to check the upper bounds or maintain a counter
 - No code change required if number of array elements are changed

```
ForEach Name In CustomerNamesArray
    Message Attention "The Customer Name is" &+ Name
End
```

Use Graphical Forms to Consolidate Ispecs

- Fixed Mode Ispec form data is limited to under 1,920 bytes
- Transactions with large data requirements become complex multi-Ispec dialogues
- Graphical Ispec forms have a much larger size limit of around 65 KB
- Enables multi-Ispec dialogues to be replaced with a single transaction
 - Must not have a Fixed Mode form
 - No terminal emulator access
 - Code will be reduced in volume and complexity
 - No dialogue management requirement, just business functionality
 - Reduced transaction processing overhead in Runtime environment
 - Much better end user experience

Use the ReEIDor Refactoring Utility

- Standalone command line utility for reformatting and refactoring of LDL+ logic on models migrated from EAE
- Minimize Insertable Substitution Strings
 - Analyses each insertable and the methods that insert the insertable
 - Updates the insertable substitution strings based on two rules
 - Remove all the substitution strings that are exactly same in every Insert statement
 - Remove common qualifiers from qualified names in substitution strings

<pre>// Before Refactor Insert statements Insert IGLG (_DA1 = GRPA.GRPB.DA1 & _DA2 = GRPA.GRPB.DA2 { Insert IGLG (_DA1 = GRPX.GRPY.DA1 & _DA2 = GRPX.GRPY.DA2 { // Before Refactor Insertable IGLG logic Move 1 _DA1 Move 2 _DA2 Move 3 _DA3</pre>	<pre>MOVE "X" DA1 ADD 1 DA2 DW (DA1 = GLB.SPACES) JTO LABEL1 END ABORT R1</pre>	<pre>// After Refactor Insert statements Move "X" DA1 Add 1 DA2 DoWhen (DA1 = GLB.SPACES) JumpTo LABEL1 End Abort R1</pre>
---	---	--

Change logic layout based on specific code



Conclusion

In Conclusion ...

- Refactoring is the process of restructuring existing computer code without changing its external behaviour
- The objective of Refactoring is to improve source code
- It does not require a “big bang” approach (which may not be acceptable anyway)
 - Plan for continuous improvement over time, adopting an approach that ...
 - Assumes all new methods use parameters, return values, and are encapsulated whenever possible
 - Refactors migrated methods when they need to be modified for BAU requirements
 - Has a background method refactoring program
 - Make it an iterative process to evolve the code base over time
- May be possible to automate some bulk changes using standard or bespoke tools

**Thank
You**

