

# BP301 – Advanced Object Oriented Programming for LotusScript

---

*Bill Buchan – HADSL*

Lotusphere® 2007



**IBM**®

# Agenda

---

- **Introductions**
- Why Object Orientated Programming
- Basic OO techniques
- Advanced OO techniques
- Summary
- Questions and Answers

# Introduction

- Who am I?
  - ▶ Bill Buchan, HADSL
    - 23 years IT experience, 12 with Lotus Notes
    - Dual PCLP in v3, v4, v5, v6, v7
    - Founder and CEO of a Business Partner – HADSL – in the UK
    - I was 40 years old on Sunday. Something a lot of people thought would never happen...
- Who are you ?
  - ▶ A LotusScript developer
  - ▶ You wish to improve your code, write more efficiently, & write more robust code
  - ▶ You have used some object orientated programming techniques already.
    - Or you're a very fast learner...

# Agenda

---

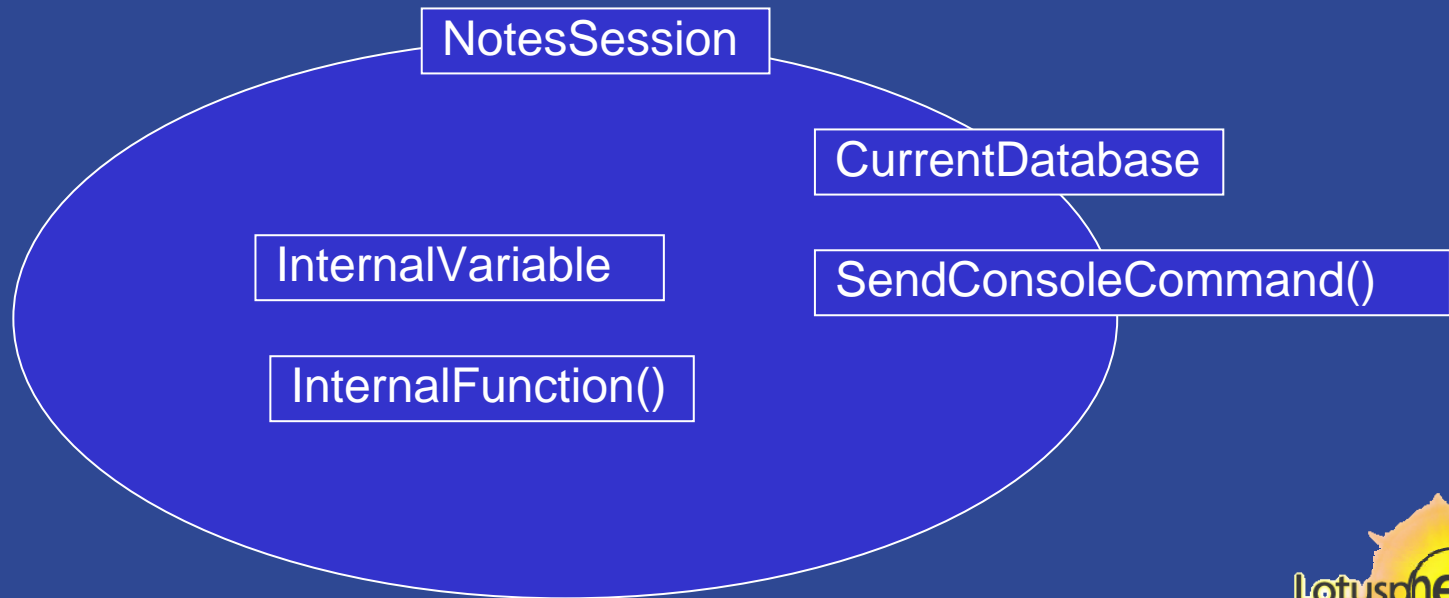
- Introductions
- **What is Object Orientated Programming?**
- Basic OO techniques
- Advanced OO techniques
- Summary
- Questions and Answers

# Why Object Orientated Programming?

- Object Orientated Programming:
  - ▶ Allows you to design higher-level objects.
  - ▶ Allows far more code-reuse.
  - ▶ Allows you to write less code.
    - And therefore leads to easier maintenance.
- Using OOP, we can componentise the manufacture of objects within our application
  - ▶ And use common techniques to deal with those objects.

# Objects – Public and Private:

- An object has to have
  - ▶ some public methods (functions/subs).
  - ▶ may have public members (variables).
- Internal code for the object need not be exposed
  - ▶ “Code Hiding” means not showing the consumer more than he needs to see.
  - ▶ Importantly – the consumer now codes to the public interface of that object.



# How to design an Object Orientated Application

- Decide which objects make sense in terms of the problem space.
  - ▶ Make the object represent logical entities within the application.
  - ▶ “Person”, “PersonCollection”, “Transaction”, etc are good examples.
- Write these object names down on separate pieces of paper
- Perform an application “walk through”
  - ▶ This will quickly show which objects need to perform particular operations, and which objects interact with other objects.
  - ▶ This helps define the “public” interface to the Object.
  - ▶ Don’t worry as to how operations within objects are performed at this stage.
- Note:
  - ▶ More sophisticated approaches exist.
  - ▶ Remember, OO programming is common in other environments.

# OOP & LotusScript

- To create a class in LotusScript, wrap code and data around “class” definitions. Such as:

```
Class Person
  Private nnName as NotesName
  Public strInternetAddress as String
  Sub new(doc as NotesDocument)
    Set nnName = doc.GetItemValue("FullName")(0)
    me.strInternetAddress = doc.getItemValue("InternetAddress")(0)
  End sub
  Public function getName
    If (me.nnName is nothing) then exit function
    getName = nnName.Abbreviated
  End function
End class
```

- And consume it with:

```
Dim P as new Person(doc)
Print "Person: " + P.getName()
```

# Constructors and Destructors

- “Sub New()” in a LotusScript Class is its “constructor”
  - ▶ It is executed automatically when the class is constructed.
  - ▶ If the class is inherited, then the parent class’s constructor is ran first.
    - This means that you need not know how the parent class constructor works
    - If you change the constructor in one class, you need not update code in any sub-classes constructors.
  - ▶ You can pass parameters to it.
  - ▶ Useful for code setup.
- “Sub Delete()” in a LotusScript class is its “destructor”
  - ▶ It is executed automatically when the object is out-of-scope, or is deleted.
  - ▶ Useful for clean-up code.
  - ▶ Very useful for disposing of critical memory objects!
    - C-API programming, COM objects, etc.

# Why does that help ?

- It helps us construct complex (and possibly interconnected) in-memory data structures
- It sites the code for that data structure alongside the data structure
  - ▶ Making maintenance easier
- We can use these complex data structures to represent far more complex algorithms.
- We can architect solutions at a far higher level, using objects (instances of these classes).
- Classes can inherit from other classes, thus aiding code re-use(dramatically).

# Where does this help us ?

---

- Complex code.
- Large applications.
- Applications that have or may change data structure frequently.
  - ▶ Object Orientated Programming helps segregate objects from each other, insulating against data or structure changes.
- Groups of applications that share common data structures.

# Examples:

- Directory Comparison tools.
  - ▶ Its far faster to read multiple directories into memory and perform comparisons between in-memory objects.
- Workflow systems
  - ▶ Where you can “hide” the complexity of the workflow from other code.
- Interfaces to External Systems
  - ▶ Where the Interface can be complex.
  - ▶ Or where you wish to support multiple interfaces.
- Where you are attempting to perform complex relational operations.

# Agenda

---

- Introductions
- Why Object Orientated Programming
- Basic OO techniques
- **Advanced Object Orientated Techniques**
- Summary
- Questions and Answers

# Lets make some assumptions

- These techniques will probably not help in very small projects
- These techniques are overkill for simple algorithms
- Where these techniques are invaluable are in
  - ▶ Large data operations
  - ▶ Relational data operations, where entire data sets have to be compared
    - For instance, directory comparison routines
  - ▶ Interface code – interfacing a Notes database to something else
    - For instance, a relational database
- Try not to be prescriptive
- Keep it simple!

# Organising Objects with Lists

- Lists. A collection construct that's always been in LotusScript.
- You can construct a memory construct where you have zero or more items in a list, and each item is referenced by a lookup value.
  - ▶ Like an in-memory "view".
- How many people know how to use lists ?
  - ▶ Resources:
    - The View – December 2006 article.
    - <http://www.billbuchan.com/web.nsf/htdocs/BBUN67JJCV.htm>
  - ▶ Can Lists work with Classes ?
    - Of course!

# Lets make a list of Person Objects

- Store Person Objects by Name:

```
Class Person
  Private nnName as NotesName
  Sub new(doc as NotesDocument)
    Set nnName = doc.FullName(0)
  End sub
  Public function getName
    If (me.nnName is nothing) then exit function
    getName = nnName.Abbreviated
  End function
End class
```

```
Dim vLookup as NotesView
...
Dim doc as NotesDocument
Set doc = vLookup.GetFirstDocument()
Dim People list as Person
While not doc is nothing
  dim P as new Person(doc)
  set People(P.getName) = P
  set doc = vLookup.GetNextDocument(doc)
wend
```

# Why is this significant ?

- A list is a very efficient (for both memory and performance).
- Consider it to be the same as a view containing documents.
  - ▶ But everything is in memory – therefore 1,000x faster.
- Because of its low memory footprint.
  - ▶ Its more efficient to have multiple lists in memory than attempt complex operations on a single list.

# Extending Classes

- Classes can be extended from other classes, and inherit code+properties.
- This leads to code-reuse.
- If more code is added to BaseClass, this is available to PersonClass

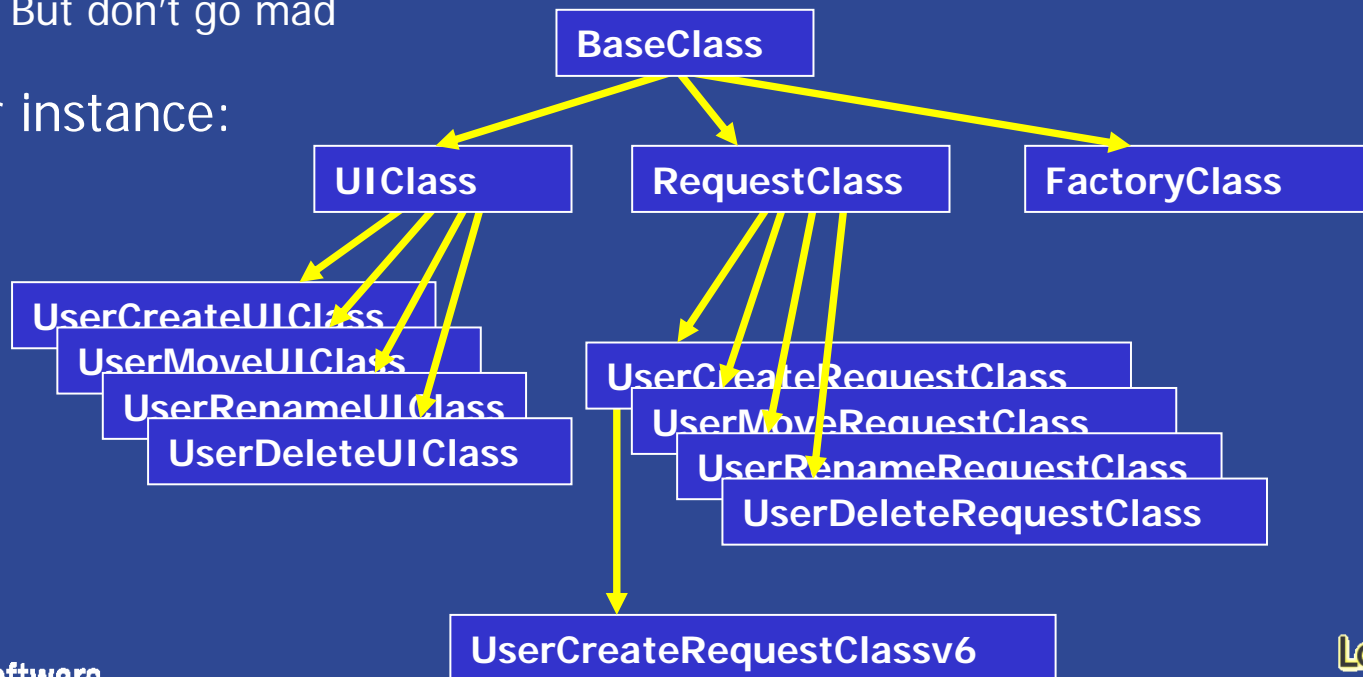
```
Class baseClass
...
Private Function log(msg as String)
...
end function
End class
```

```
Class PersonClass as BaseClass
...
sub new(doc as NotesDocument)
...
call me.log("Started")
...
end sub
End class
```

# Extending Classes...

- When one class is extended from another class
  - ▶ It inherits all code and data from that original class.
  - ▶ It can execute private methods and access private members of the parent classes.
- There is no limit to the level of inheritance
  - ▶ But don't go mad

- For instance:



# Inheritance Trees and Code Placement

- Inheriting all classes from a common root means
  - ▶ Common code can be placed high in the inheritance tree and be used by all.
    - If this code changes, its only changed in one place.
    - Lots of code reuse.
  - ▶ Examples of Infrastructure code might be:
    - Logging.
    - Persistence.
    - Error trapping.
    - Configuration.
  - ▶ New code can be placed at the relevant level in the tree.
  - ▶ Each individual class can be kept to a manageable size:
    - No 1,300 line functions!
    - Architectural decisions can be followed - No "Design Drift".

# Inheritance

- Note that in large systems:
  - ▶ You may end up with Classes that are not used by the application itself, but are placeholders for inheritance points and inherited code/data.
    - This is not a bad thing! It keeps the design “clean”, and keeps infrastructure code out of your business objects!
- You may wish to use the Factory design pattern
  - ▶ Where one class is responsible for creating instances of other classes.
  - ▶ It’s a useful point for deciding whether an object is a Singleton or not.
  - ▶ Its also a place to keep lists of all classes that are created:
    - Useful for NSD-Style error trapping analysis!

# Version/Platform Specific code insulation

- Class inheritance allows you to insulate code from version or platform differences
- For instance, a class may be written using v5 features
  - ▶ A v6 version of the class can be created which inherits from the v5 class, but only overrides the version-specific code
- To make this simple
  - ▶ Group version specific code into particular functions which can be easily identified and overridden.

# Version/Platform Specific code insulation - Example

```
Class personClass
...
Private Function createUser(userName as String) as integer
...
end function
End class
```

```
Class PersonClassV6 as PersonClass
Private Function createUser(userName as String) as integer
' place v6 specific code here
end function
End class
```

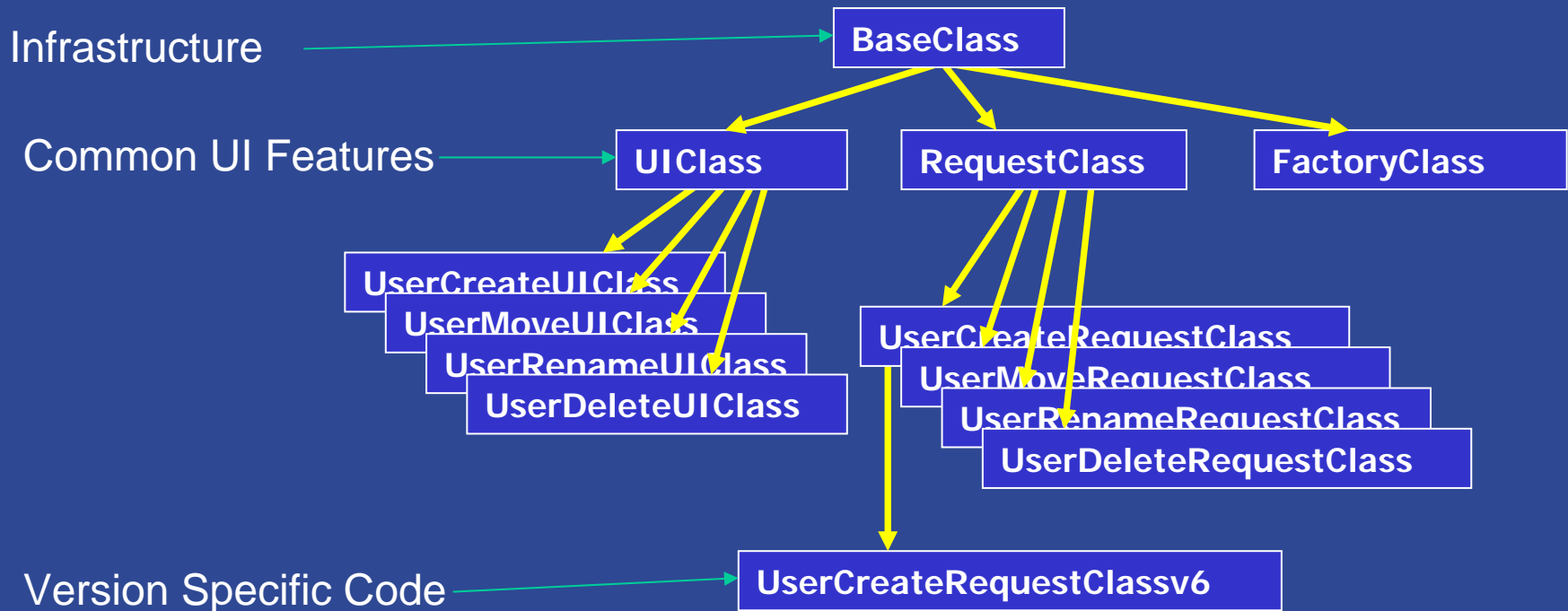
```
Dim S as new NotesSession
Dim P as variant
If S.NotesBuildVersion < 170 then
' Its v5
set P = new PersonClass()
Else
' V6 and above.
set P = new PersonClassV6()
End if

' Use P as normal from now on.
```

Note that the V6 class usually ONLY contains functions that have been overridden. It may be very small.

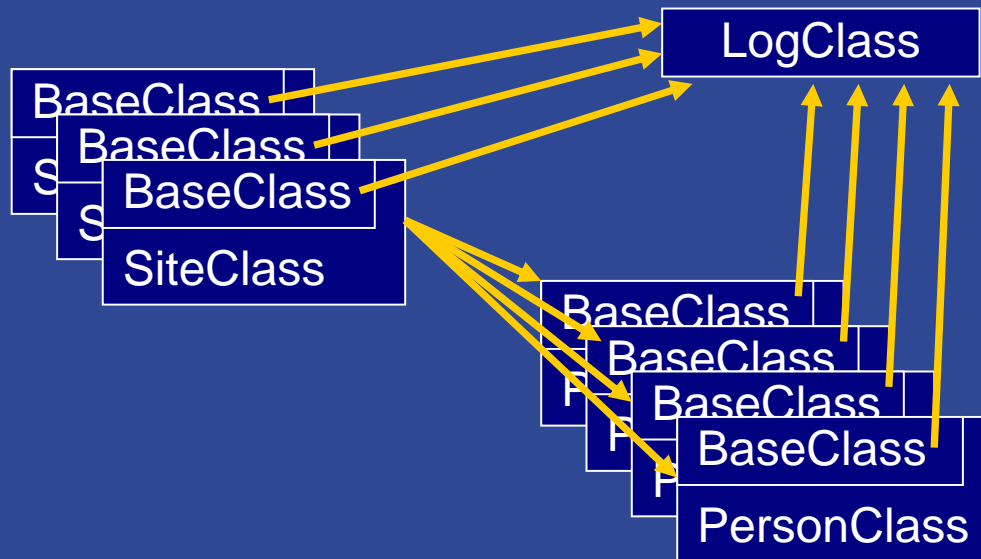
1. Good Code Reuse.
2. Business Logic remains in one place.

# Inheritance Trees and Code Placement - Example



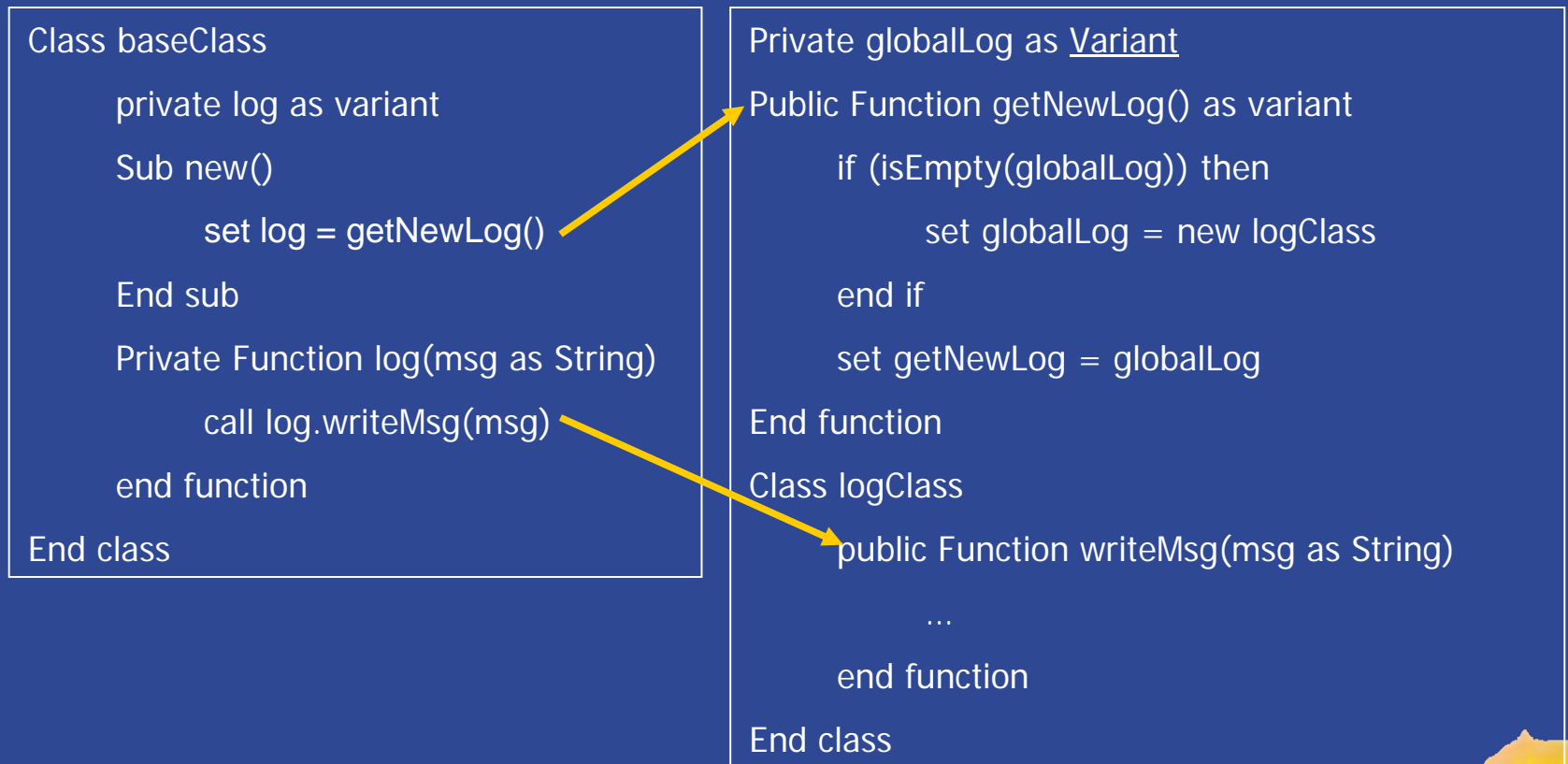
# BaseClass Specification

- We would like to put some form of logging into our Baseclass
- However, lots of other objects will inherit from BaseClass.
- Our Baseclass cannot create a new NotesAgentLog class for each instance.
  - ▶ We need to only create ONE single instance of our NotesAgentLog class.



# The Singleton Design Pattern

- “Singleton” is a design pattern which ensures that ONLY one copy of an object exists at one time:



# Infrastructure Code: Configuration

- Typically, a large application will have lots of configuration values
  - ▶ Hard coded design decisions can be exposed as configuration variables
  - ▶ This makes the application far more useful.
    - Allows the application to be used in ways not originally envisaged at design time, without code changes.
  - ▶ Flexibility – whilst a positive thing – does come at a price
    - Configuration variables have to be exposed, checked and read at Run-time.
- I could have a single configuration profile document and a single configuration object
  - ▶ BaseClass could then link to that as a Singleton at run time.
  - ▶ Making all configuration options available to all objects in memory.

# Infrastructure Code: Persistence

- “Persistence” is the ability of an object to make itself persistent between executions
- We could easily add a NotesDocument, and methods to manipulate that document available in the BaseClass
- However, the BaseClass cannot know the layout of objects inherited from it
  - ▶ Therefore objects that inherit from BaseClass must have their own persistence routines coded which make use of the hooks from BaseClass.
- An object need not be persistent
  - ▶ Nor does it need to “save” itself between runs – it might only “read” itself from a profile document.

# Infrastructure code: Error Trapping

- BaseClass is a good place to put a single error trapping routine:

```
Function RaiseError() as integer
  Dim thisType As String, es as String
  thisType = Typename(Me)

  ' Not a class, use the calling module instead
  If (thisType = "") Then thisType = Getthreadinfo(11)

  es = thisType & "::" & Getthreadinfo(10) & ":: "

  If (Err = 0) Then
    es = es + "Manually raised an error"
  Else
    es = es + "Run time error: (" + Trim(Str(Err)) + ") " + _
      Error$ + " at line: " + Trim(Str(Erl))
  End If

  Print es
end function
```

```
' calling code...
...
ExitFunction:
  exit function
errorhandler:
  Call RaiseError()
  resume exitFunction
end function
```

# How does all this work ?

---

- Since all classes inherit (directly or indirectly) from our BaseClass:
  - ▶ All have common infrastructure elements.
    - Logging, Persistence, Error Trapping, Configuration.
  - ▶ All defined in one single place.
- All new classes can then focus on business logic.

# Late Binding

---

- Late Binding is a method where the type of an object is decided at Run-Time.
- In LotusScript this is normally achieved using the Variant data type
- This may be useful where you have to perform the same operation on number of objects of roughly the same type.

## Late Binding Example:

```
' calling code.  
dim dc as NotesDocumentCollection  
Set dc = dbThis.UnprocessedDocuments  
Call ProcessDocs(dc)
```

```
' calling code.  
dim vLookup as NotesView  
Set vLookup = dbThis.GetView("($All)")  
Call ProcessDocs(vLookup)
```

```
Function ProcessDocs(coll as Variant) as integer  
    dim doc as NotesDocument  
    Set doc = coll.getFirstDocument  
    While (not doc is nothing)  
        ' Do something with the document  
        set doc = coll.getNextDocument(doc)  
    wend  
end function
```

# Late Binding Summary

- Late Binding in LotusScript:
  - ▶ Uses Variants.
  - ▶ Relies on the SIGNATURE of the passed object to work, not its type.
    - (Other languages would force you to have a common inheritance point)
  - ▶ If that signature is not available, this results in a Run-time error:
    - So – more testing!
- Very useful for “framework” processing code
  - ▶ A framework which will process objects exposing a common set of methods or properties.

# Architecturally:

---

- Collect data together in meaningful small objects
  - ▶ Person, Purchase Orders, etc.
- Use collection objects to group these small objects
- Use generic classes to perform business operations on these collections.
- Write out the results at the end.
  - ▶ Good “defensive coding”.

## Lets put all this together:

- Our application compares two Notes Directories
  - ▶ Identifies people who are not in both databases
- The Application:
  1. Reads in all person documents from both directories into PersonClass objects.
  2. Compares the list of PersonClass objects in both directions.
  3. Creates a list of PersonClass objects that is of "interest".
  4. Writes these out to the local database for further processing.
- ▶ This would be difficult to do using NotesDocuments and Arrays.

# Demo Application

---

- Demonstration of Advanced Object Orientated Techniques.

# Agenda

---

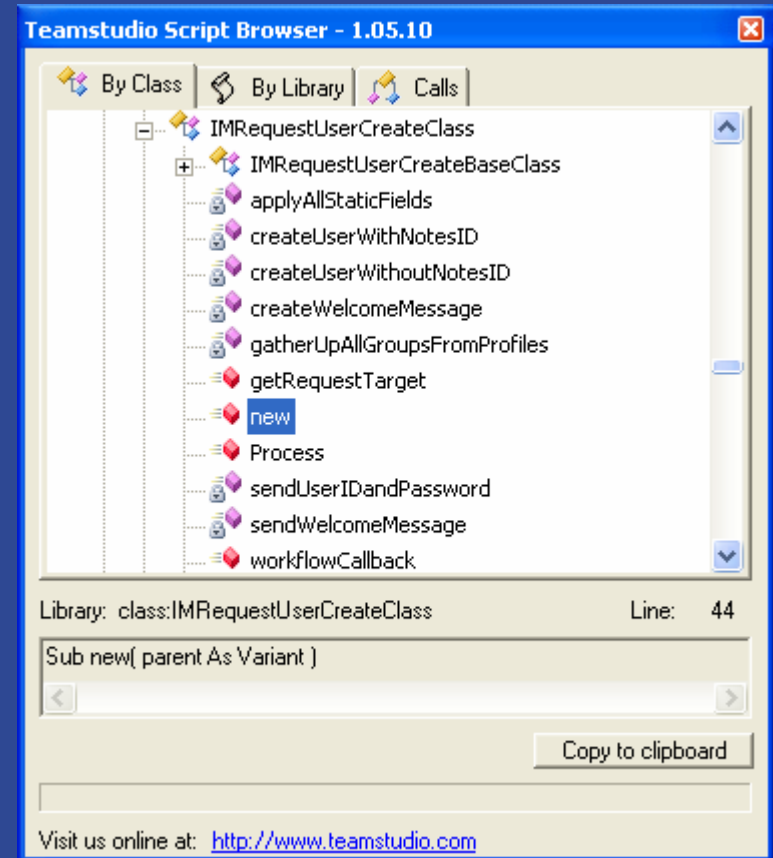
- Introduction
- Why Object Orientated Programming
- Basic OO techniques
- Advanced OO Techniques
- **OO Pitfalls**
- Summary
- Questions and Answers

# Editing Code

- All Class definitions are placed in the Declarations Section
- So the left-hand LotusScript Editor is useless.
  - ▶ Ouch!
- TeamStudio to the Rescue!
  - ▶ <http://www.teamstudio.com/support/scriptbrowser.html>
  - ▶ Download the Class Browser.
  - ▶ Its Free!
  - ▶ Allows you to see the structure, and jump to any piece of code.
  - ▶ Many thanks to Craig Schumann of TeamStudio for this.

# Editing Code

- This is a screenshot
  - ▶ From a 400+ Script Library database.
  - ▶ 625 LotusScript Classes.
  - ▶ 130,000 lines of code.



## “Error loading USE or USELSX ” Or “Type Mismatch in %F”

- This indicates that some parent class in the inheritance tree has changed its public signature.
- All dependant classes need to be recompiled.
  - ▶ “Tools, Recompile All LotusScript” is your friend.
- This is not restricted to LotusScript
  - ▶ However, other IDE’s – Eclipse, RAD – will recompile all other dependant code automatically.

# Memory Usage:

- Its very tempting to store NotesDocuments within Objects
  - ▶ However, this will use a lot of memory.
  - ▶ This will hold a lot of documents open on the server.
  
- If you need to perform comparisons on large numbers of documents:
  1. Read the relevant parts of the document into an object.
  2. Store the document NoteID or UNID.
  3. Close the document.
  4. Perform the comparison.
  5. If you need to return to the document, open by NoteID or UNID.
    - ▶ Performance wise, this is very quick.

# Class Loading Performance

- As the number of dependant classes in a system starts to exceed 20 or 30, the load-time for any system which incorporates these classes climbs exponentially.
  - ▶ Example. 120+ classes – 145 seconds to load!
- The workaround is to use Dynamic Loading.
  - ▶ Dynamic loading allows you to load a class at run-time based on data, instead of using the "Use <classLibrary>" definition.
  - ▶ It means that all classes are now Variants, and late-bound - More run-time errors.
  - ▶ outlined in Redbook: "Performance Considerations for Domino Applications"
    - SG24-5602
    - Buried in Appendix B, Page 243

# Dynamic Loading: ClassFactory

```
Class PersonClassFactory
  Public Function produce As Variant
    Set produce = New PersonClass()
  End Function
End Class
```

```
Class PersonClass
  ...
End class
```

The 'Factory' Version of this class exists only to create a new instance of the main class

There are no constructor arguments being passed to class 'PersonClass'. This will have to be changed if you choose to add constructor arguments.

# Dynamic Loading Example: CreateOtherClass

```
Private P as variant      ' This HAS to be global!

Function createOtherClass (_
    scriptName as String, _
    className as String) as variant

    Dim exString as String

    exString = | Use " | & scriptName & | "
                sub initialize
                    Set P = new | & className & |Factory &
                end sub |
    Execute exString      ' Now run the code in exString

    Set createOtherClass = P.produce()

End sub
```

This string contains  
LotusScript code!

# Dynamic Loading Example: Execution

Sub DynamicLoadingExample

```
dim myClass as variant
```

```
set myClass = createotherClass("class:PersonClass","PersonClass")
```

End sub

MyClass now contains an instance of Class "PersonClass" from script library "Class:PersonClass"

# Agenda

---

- Introduction
- Why Object Orientated Programming
- Basic OO techniques
- Advanced OO Techniques
- OO Pitfalls
- **Summary**
- Questions and Answers

# Summary

- Object Orientated Programming in LotusScript:
  - ▶ Is easy.
  - ▶ Allows far more code reuse.
  - ▶ Allows you to focus on the design.
- Developing Large Systems using Object Orientated Techniques
  - ▶ Is different.
  - ▶ Is far simpler.
  - ▶ Allows much more code re-use.
  - ▶ Allows you to write more consistent, more robust code.
- Example: Our product:
  - ▶ Has over 130,000 lines of LotusScript.
  - ▶ Supports 40+ transactions.
  - ▶ Supports multi-platform, multi-version transactions.
  - ▶ Would be impossible to write without OO techniques.

# Some Resources

- Notes Designer help: look for "class statement", "custom classes", "derived classes" ... and more
- LotusScript Programmer's Guide, Developer Works  
<http://www-10.lotus.com/ldd/notesua.nsf>)
- Article on Developer Works (Archive of Iris Today):  
Bruce Perry: Using the object oriented features of LotusScript (Oct. 1, 2001)  
[http://www-128.ibm.com/developerworks/lotus/library/ls-object\\_oriented\\_LotusScript/](http://www-128.ibm.com/developerworks/lotus/library/ls-object_oriented_LotusScript/)
- Article "Creating Custom Classes in LS" by Johan Känngård  
<http://dev.kanngard.net/dev/home.nsf/ArticlesByTitle/LSClass.html>
- Teamstudio ScriptBrowser: <http://blogs.teamstudio.com>

# Related Sessions at Lotusphere 2007

- BP507 Leveraging the Power of Object Oriented Programming in LotusScript  
Jens-B Augustini & Bill Buchan
- AD506 Intermediate LotusScript  
John Dillon
- BP308 Leverage DXL and OOP to Build Powerful Tools  
Mikkel Heisterberg
- AD504 What's New in the IBM Lotus Domino Objects?  
James Cooper

# Agenda

---

- Introductions
- Why Object Orientated Programming
- Basic OO techniques
- An Object Orientated Project
- Encapsulating Classes
- Extending Classes
- Summary
- **Questions and Answers**

# Questions and Answers

- Limited Time for Questions
  - ▶ I'm available in the speaker room immediately after this session.
- This was BP301 – Advanced Object Oriented Programming for LotusScript
- I am Bill Buchan:
  - ▶ <http://www.billbuchan.com>
  - ▶ <http://www.hadsl.com>
- Remember to fill in your Evaluations.
  - ▶ This is how YOU influence Lotusphere 2008!

© IBM Corporation 2007. All Rights Reserved.

The workshops, sessions and materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. Nothing contained in this presentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in this presentation may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

IBM, the IBM logo, Lotus, Lotus Notes, Notes, Domino, Domino.Doc, Domino Designer, Lotus Enterprise Integrator, Lotus Workflow, Lotusphere, QuickPlace, Sametime, WebSphere, Workplace, Workplace Forms, Workplace Managed Client, Workplace Web Content Management, AIX, AS/400, DB2, DB2 Universal Database, developerWorks, eServer, EasySync, i5/OS, IBM Virtual Innovation Center, iSeries, OS/400, Passport Advantage, PartnerWorld, Rational, Redbooks, Software as Services, System z, Tivoli, xSeries, z/OS and zSeries are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torbvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

All references to Acme, Renovations and Zeta Bank refer to a fictitious company and are used for illustration purposes only.